REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	September 1994	Master's Thesis

4. TITLE AND SUBTITLE
Optimizing Safe Motion for Autonomous Vehicles (u)

5. FUNDING NUMBERS

6. AUTHOR(S)
Shirasaka, Masahide

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey, CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/ MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING/ MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the United States Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

There are two goals for autonomous vehicle navigation planning: shortest path and safe path. These goals are often in conflict; path safety is more important. Safety of the autonomous vehicle's navigation is determined by the clearances between the vehicle and obstacles. Because a Voronoi boundary is the set of points locally maximizing the clearance from obstacles, safety is maximized on it. Therefore Voronoi Diagrams are suitable for motion planning of autonomous vehicles.

We use the derivative of curvature k of the vehicle motion (dk/ds) as the only control variable for the vehicle where s is the length along the vehicle trajectory. Previous motion planning of the autonomous mobile robot Yamabico-11 at Naval Postgraduate School used a path tracking method. Before the mission began the vehicle was given a track to follow; motion planning consisted of calculating the point on the track closest to the vehicle and calculating dk/ds then steering the vehicle to get onto track.

We propose a method of planning safe motions of the vehicle to calculate optimal dk/ds at each point directly from the information of the world without calculating the track to follow. This safe navigation algorithm is fundamentally different from the path tracking using a path specification. Additionally motion planning is simpler and faster than the path tracking method.

The effectiveness of this steering function for vehicle motion control is demonstrated by algorithmic simulation and by use on the autonomous mobile robot Yamabico 11 at the Naval Postgraduate School.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Optimizing Safe Motion for Autonomous Vehicles	84
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	Unlimited

Approved for public release; distribution is unlimited.

OPTIMIZING SAFE MOTION

FOR

AUTONOMOUS VEHICLES

by

Masahide Shirasaka
Lieutenant Junior Grade, Japan Maritime Self-Defense Force
B.S., Japan Defense Academy , 1989

Submitted in partial fulfillment
of the requirements for the degree

MASTER OF SCIENCE IN OPERATIONS RESEARCH

from the

NAVAL POSTGRADUATE SCHOOL
September 1994

ABSTRACT

There are two goals for autonomous vehicle navigation planning: shortest path and safe path. These goals are often in conflict; path safety is more important. Safety of autonomous vehicle navigation is determined by the clearance between the vehicle and obstacles. Because a Voronoi boundary is the set of points locally maximizing the clearance from obstacles, safety is maximized on it. Therefore, Voronoi Diagrams are suitable for motion planning of autonomous vehicles.

We use the derivative of curvature κ of the vehicle motion ($d\kappa/ds$) as the only control variable for the vehicle, where s is the length along the vehicle trajectory. Previous motion planning of the autonomous mobile robot Yamabico-11 at the Naval Postgraduate School used a path tracking method. Before the mission began the vehicle was given a track to follow; motion planning consisted of calculating the point on the track closest to the vehicle and calculating $d\kappa/ds$ then steering the vehicle to get onto the track.

We propose a method of planning safe motions of the vehicle to calculate optimal $d\kappa/ds$ at each point directly from the information of the world without calculating the track to follow. This safe navigation algorithm is fundamentally different from path tracking using a path specification. Additionally, motion planning is simpler and faster than the path tracking method.

The effectiveness of this steering function for vehicle motion control is demonstrated by algorithmic simulation and by use on the autonomous mobile robot Yamabico 11 at the Naval Postgraduate School.

THESIS DISCLAIMER

The reader is cautioned that computer programs developed in this research may not have been exercised for all cases of interest. While every effort has been made, within the time available, to ensure that the programs are free of computational and logic errors, they cannot be considered validated. Any application of these programs without additional verification is at the risk of the user.

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGMENTS

The author greatly appreciates those who advised, helped, cooperated with and encouraged him. Without their efforts, this thesis would never have been completed. In particular, the author wishes to thank his Principal Advisor, Dr. Yutaka Kanayama for his unwavering support and guidance throughout this entire experience.

EXECUTIVE SUMMARY

There are two goals for autonomous vehicle navigation planning: shortest path and safe path. These goals are often in conflict; path safety is more important. Safety of autonomous vehicle navigation is determined by the clearances between the vehicle and obstacles. Because a Voronoi boundary is the set of points locally maximizing the clearance from obstacles, safety is maximized on it. Therefore, Voronoi Diagrams are suitable for motion planning of autonomous vehicles.

We use the derivative of curvature κ of the vehicle motion ($d\kappa/ds$) as the only control variable for the vehicle, where s is the length along the vehicle trajectory. Previous motion planning of the autonomous mobile robot Yamabico-11 at the Naval Postgraduate School used a path tracking method. Before the mission began the vehicle was given a track to follow; motion planning consisted of calculating the point on the track closest to the vehicle and calculating $d\kappa/ds$ then steering the vehicle to get onto the track. The safest path through the world navigated by the vehicle is the set of points locally maximizing the clearance from obstacles. This path is represented by the Voronoi diagram. To achieve the path tracking method it is necessary to calculate the Voronoi boundary which consists of line segments and parabolic arcs. If the world is large, it is complicated and inefficient to calculate every Voronoi boundary of this world. It is better to calculate optimal $d\kappa/ds$ at each point directly from the information of the world without calculating the track to follow.

When the objects are two points, two lines, or one point and one line, we can safely navigate the vehicle to achieve equal clearances from these objects. The

motion of the vehicle is optimized at each point directly from the information of the obstacles near the vehicle. After calculating $d\kappa/ds$, the vehicle follows the Voronoi boundaries defined by the environment.

Unlike the path tracking method, this method can be applied to avoid moving objects since we calculate the optimal motion of the vehicle at each point directly from the information of the world. Additionally, motion control is simpler and faster than in the path tracking method.

The effectiveness of this steering function for vehicle motion control is demonstrated by algorithmic simulation and by use on the autonomous mobile robot Yamabico 11 at the Naval Postgraduate School. It has precise knowledge of its location in a given environment using its sonar system. The robot is programmed by the high level mobile robot language called MML (Model-based Mobile robot Language) written in the C language.

I. INTRODUCTION

A. BACKGROUND

There are two goals for planning autonomous vehicle navigation planning: shortest path and safe path. These goals are often in conflict; path safety is more important. Safety of autonomous vehicle navigation is determined by the clearance between the vehicle and obstacles. Because a Voronoi boundary is the set of points locally maximizing the clearance from obstacles, safety is maximized on it. Therefore, Voronoi Diagrams are suitable for motion planning of autonomous vehicles.

B. OVERVIEW

We use the derivative of curvature κ of the vehicle motion ($d\kappa/ds$) as the only control variable for the vehicle, where s is the length along the vehicle trajectory. Previous motion planning of the autonomous mobile robot Yamabico-11 at the Naval Postgraduate School used a path tracking method [Ref. 2]. Before the mission began the vehicle was given a track to follow; motion planning consisted of calculating the point on the track closest to the vehicle and calculating $d\kappa/ds$ then steering the vehicle to get onto the track. The safest path through the world navigated by the vehicle is the set of points locally maximizing the clearance from obstacles. This path is represented by the Voronoi diagram. To achieve the path tracking method it is necessary to calculate the Voronoi boundary which consists of line segments and parabolic arcs. If the world is large, it is complicated and inefficient to calculate every Voronoi boundary of this

world. It is better to calculate optimal $d\kappa/ds$ at each point directly from the information of the world without calculating the track to follow.

This safe navigation algorithm is fundamentally different from path tracking using a path specification. Additionally, motion planning is simpler and faster than in the path tracking method.

The effectiveness of this steering function for vehicle motion control is demonstrated by algorithmic simulation and by use on the autonomous mobile robot Yamabico 11 at the Naval Postgraduate School. It has precise knowledge of its location in a given environment using its sonar system. The robot is programmed by the high level mobile robot language called MML (Model-based Mobile robot Language) written in the C language [Ref. 3].

II. PROBLEM STATEMENT

The problems addressed are as follows:

1. How to navigate the robot safely to achieve the same clearance from two points.

2. How to navigate the robot safely to achieve the same clearance from two lines.

3. How to navigate the robot safely to achieve the same clearance from one point and one line.

4. How to execute the safest path by a real robot.

Safety is one of the important attributes for autonomous vehicle navigation planning. Safety is determined by the clearance between the vehicle and objects. Assume the vehicle is a point object and $W(p)$ is the clearance of the point p. The clearance of a path represents its safety. If the clearance is small, the path is dangerous because it is close to the object and if it is larger, the path is safer. The clearance of a path Π is defined as:

$$W(\Pi) = \underset{P \in \Pi}{Min} W(P) \tag{2.1}$$

Where Π is the path of the vehicle from start S to goal G.

We want to find the path Π_0 such that $W(\Pi_0)$ is the maximum among all possible paths (Figure 2.1). To maximize the clearance $W(\Pi)$, we will take the Voronoi boundary as the path of the vehicle.

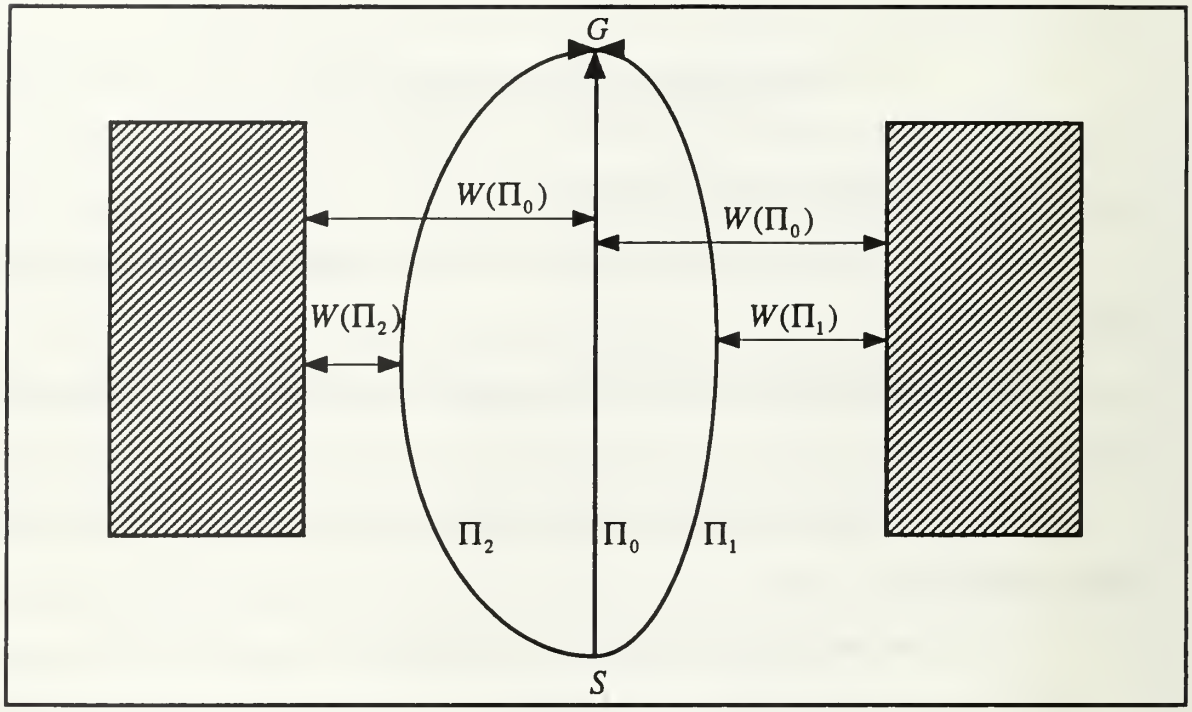

Figure 2.1 : Safety Path

III. VORONOI DIAGRAM

A. DEFINITIONS

Assume that there is an object o in a plane. It might be a point, a line, a line segment, a polygon, or other closed sets of points. If a world W has more than one object, a *Voronoi region* of an object o_i in the world is defined as

$$V(o_i) = [P | \forall j[i \neq j \rightarrow \{dist(p,o_i) \leq dist(p,o_j)\}]] \qquad (3.1)$$

Where p is a point which is not inside o_i and $dist(p,o_i)$ is the minimum distance between p and o_i.

The set of all the *Voronoi regions* is called the *Voronoi diagram* of a world. The boundaries of *Voronoi regions* are *Voronoi boundaries*. The Voronoi diagram of a geometric world typically consists of lines, rays, line segments, and parabolic arcs.

B. VORONOI DIAGRAM OF TWO POINTS

In the case that a world consists of two distinct points, p_1 and p_2, its Voronoi boundary is the bisector of those two points which generate two Voronoi regions $V(p_1)$ and $V(p_2)$ (Figure 3.1).

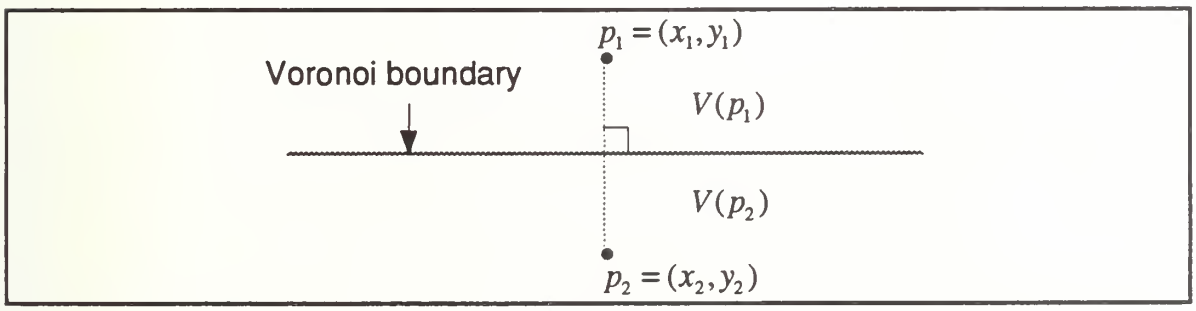

Figure 3.1 : Voronoi Diagram of Two Points

C. VORONOI DIAGRAM OF TWO LINES

In the case that a world consists of two lines L_1 and L_2 which are not parallel to each other, their Voronoi boundaries consist of the bisectors of two lines which generate eight Voronoi regions $V(L_{11})$, $V(L_{12})$, $V(L_{13})$, $V(L_{14})$, $V(L_{21})$, $V(L_{22})$, $V(L_{23})$ and $V(L_{24})$ (Figure 3.2).

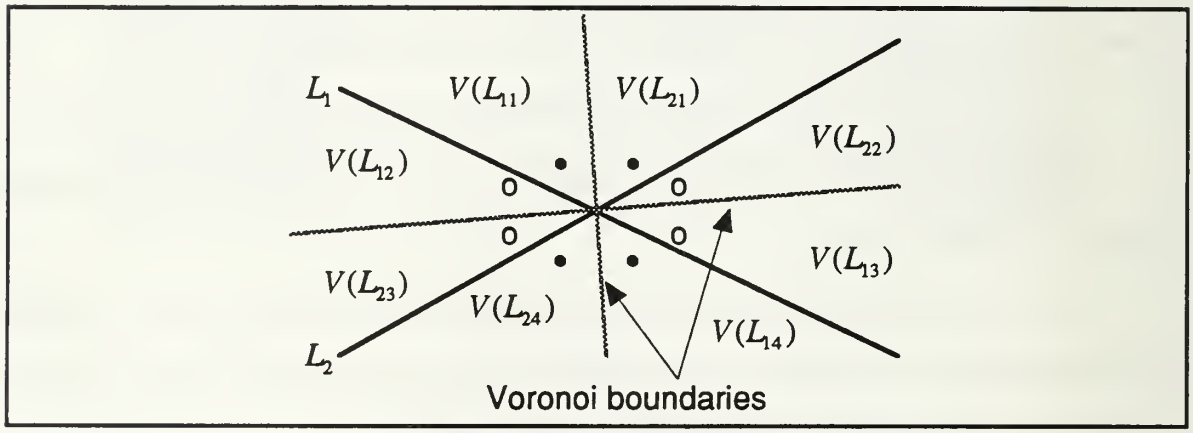

Figure 3.2 : Voronoi Diagram of Two Lines (Not Parallel)

D. VORONOI DIAGRAM OF TWO PARALLEL LINES

In the case that a world consists of two parallel lines L_1 and L_2, their Voronoi boundary is one line which is parallel to L_1 and L_2 which generates four Voronoi regions $V(L_{11})$, $V(L_{12})$, $V(L_{21})$ and $V(L_{22})$, (Figure 3.3).

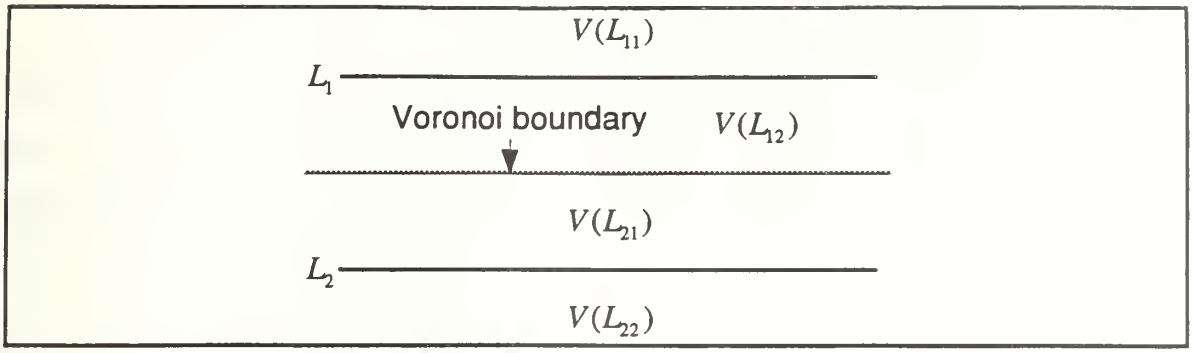

Figure 3.3 : Voronoi Diagram of Two Parallel Lines

E. VORONOI DIAGRAM OF A LINE SEGMENT

In the case that a world consists of one closed line segment $\overline{p_1p_2}$, we treat it as a union of three objects : two end points p_1, p_2 and an open line segment $e = \overline{p_1p_2}$. A closed line segment includes both endpoints, but an open line segment does not. Therefore, its Voronoi boundaries consists of two lines which generate four Voronoi regions $V(p_1)$, $V(p_2)$, $V(e_1)$ and $V(e_2)$. (Figure 3.4)

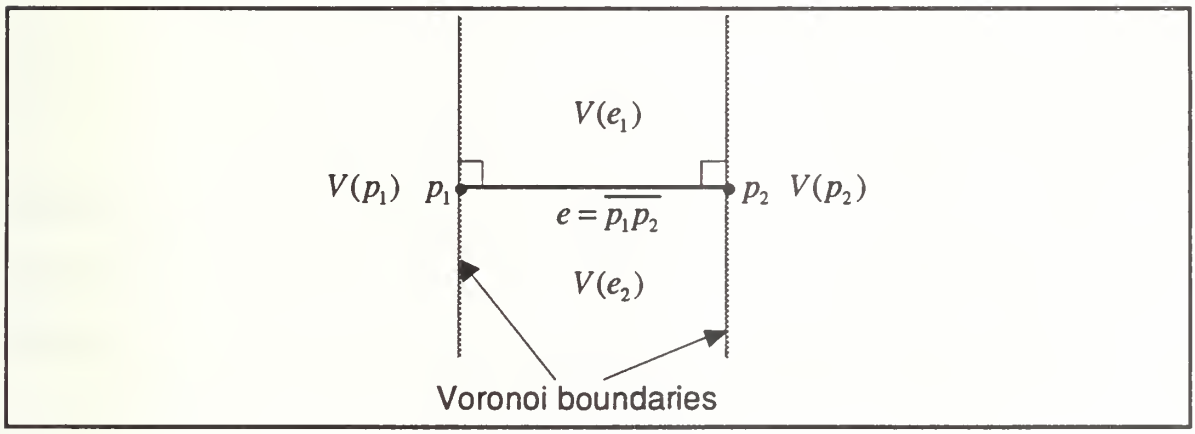

Figure 3.4 Voronoi Diagram of A Line Segment

F. VORONOI DIAGRAM OF A POINT AND A LINE

In the case that a world consists of a point p_f and a line q_0, its Voronoi boundary is a parabola which generates three Voronoi regions $V(p_f)$, $V(e_1)$, and $V(e_2)$. The point p_f is the *focus* and the line q_0 is the *directrix* of the parabola (Figure 3.5).

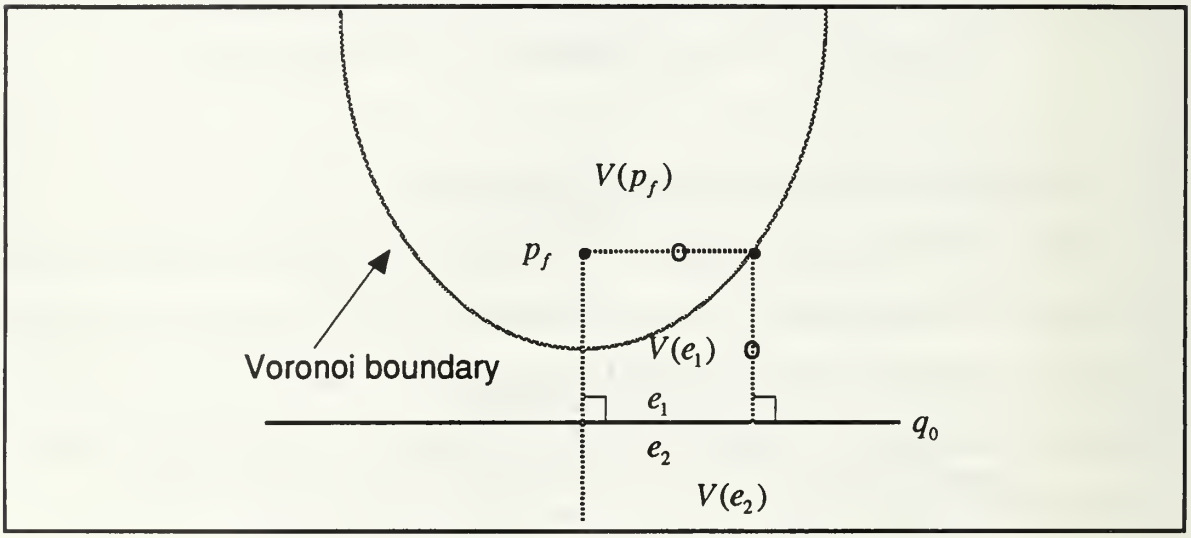

Figure 3.5 : Voronoi Diagram of A Point and A Line

G. VORONOI DIAGRAM OF A POINT AND TWO LINES

In the case that a world consists of a point p that is between two parallel lines L_1 and L_2, their Voronoi boundaries are two parabolas (focus p, directrix L_1 and focus p, directrix L_2) and the bisector of the line L_1 and L_2 which generate five Voronoi regions $V(p)$, $V(e_1)$, $V(e_2)$, $V(e_3)$ and $V(e_4)$ (Figures 3.6 and 3.7).

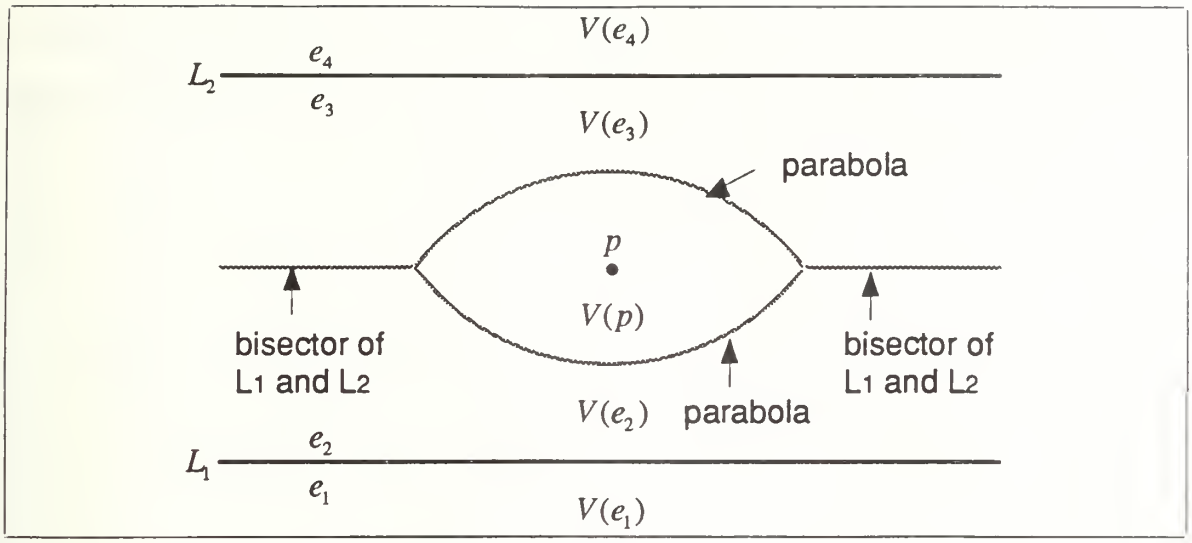

Fig 3.6 : Voronoi Diagram of A Center Point and Two Lines

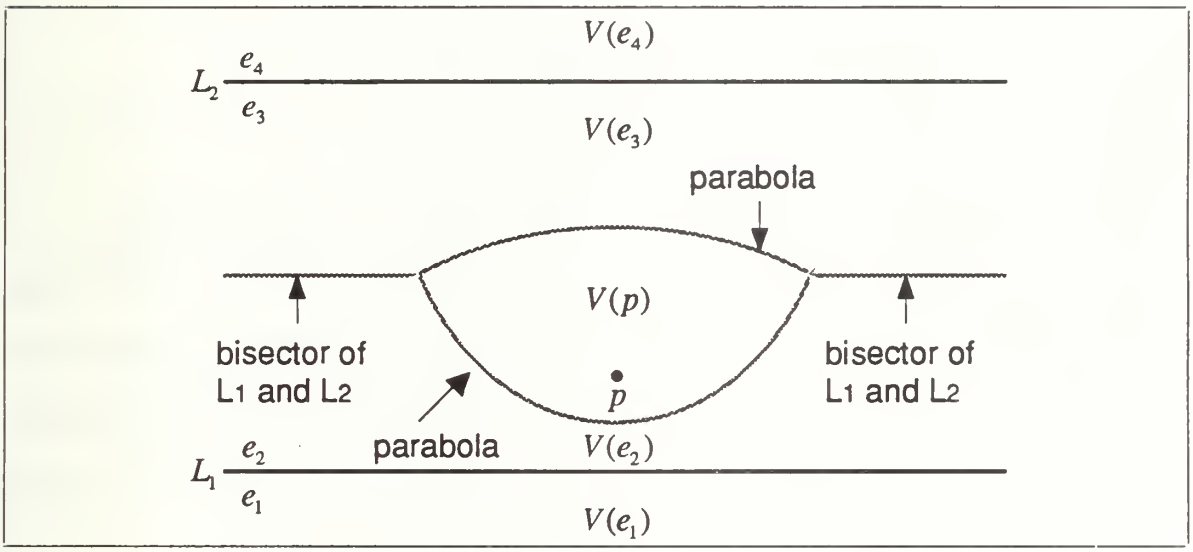

Figure 3.7 : Voronoi Diagram of A Point and Two Lines

H. VORONOI DIAGRAM OF LINE AND A RAY

Assume a ray is a kind of line which has only one end point p. In the case that a world consists of a line L_1 and a ray L_2, which is orthogonal to the line L_1,

their Voronoi boundaries are a line segment $\overline{p_1p_2}$, and two bisectors of the line L_1 and the ray L_2, and a parabola (focus p and directrix L_1), which generates five Voronoi regions $V(p)$, $V(e_1)$, $V(e_2)$, $V(e_3)$ and $V(e_4)$ (Figure 3.8).

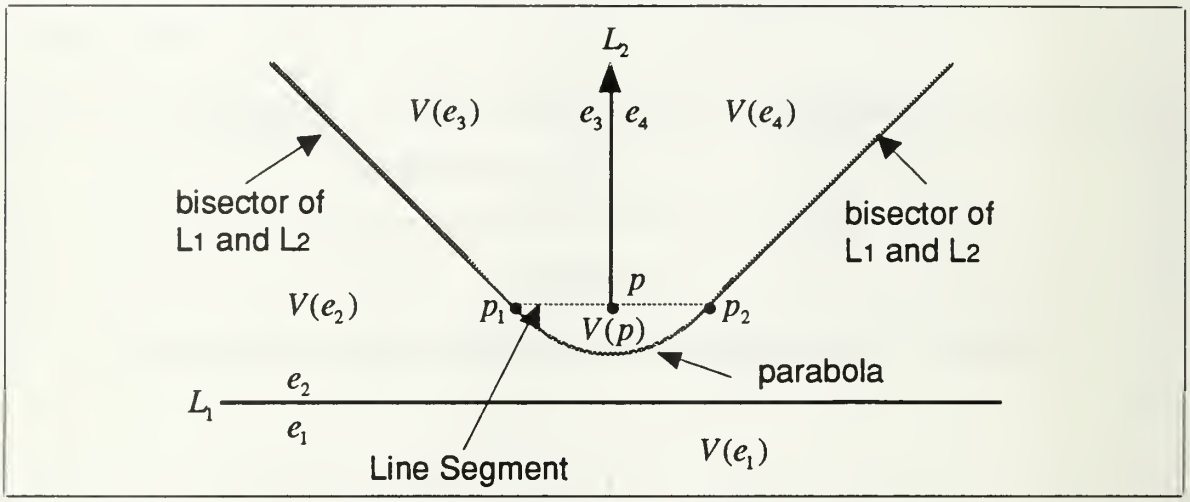

Figure 3.8 : Voronoi Diagram of Line and Ray

I VORONOI DIAGRAM OF TWO LINES AND A LINE SEGMENT

In the case that a world consists of two parallel lines L_1, L_2 and a closed line segment $\overline{p_1p_2}$, which is parallel to lines L_1 and L_2, their Voronoi boundaries are bisectors of the line L_1 and L_2, the bisector of the line L_1 and the line segment $\overline{p_1p_2}$, the bisector of the line L_2 and the line segment $\overline{p_1p_2}$, two closed line segment $\overline{p_3p_4}$ and $\overline{p_5p_6}$ and parabolas (Figures 3.9 and 3.10).

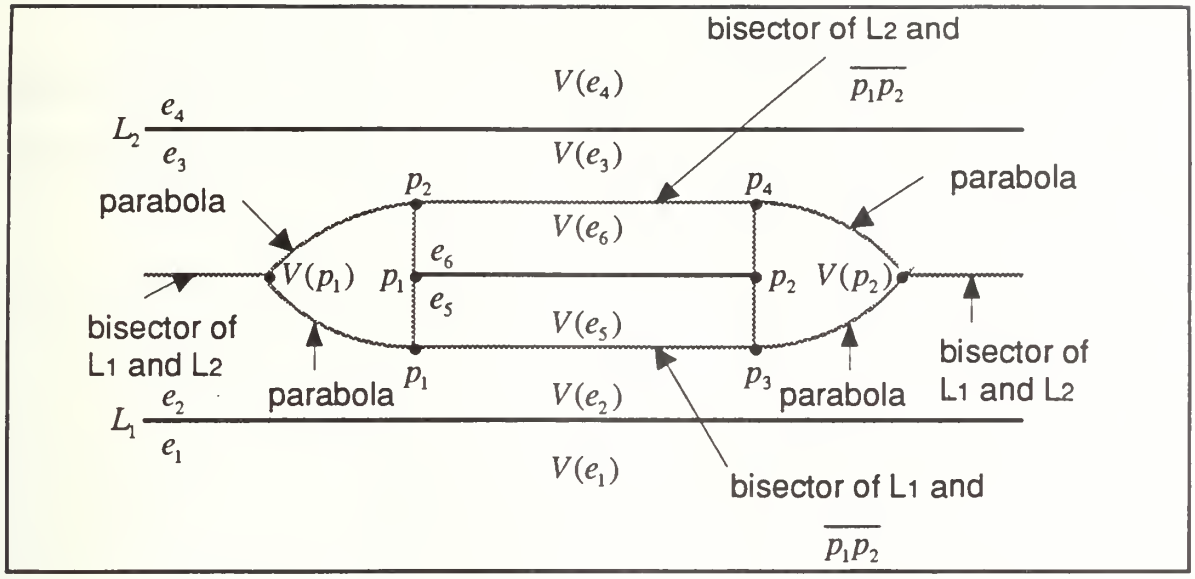

Figure 3.9 : Voronoi Diagram of Two Lines and A Center Line Segment

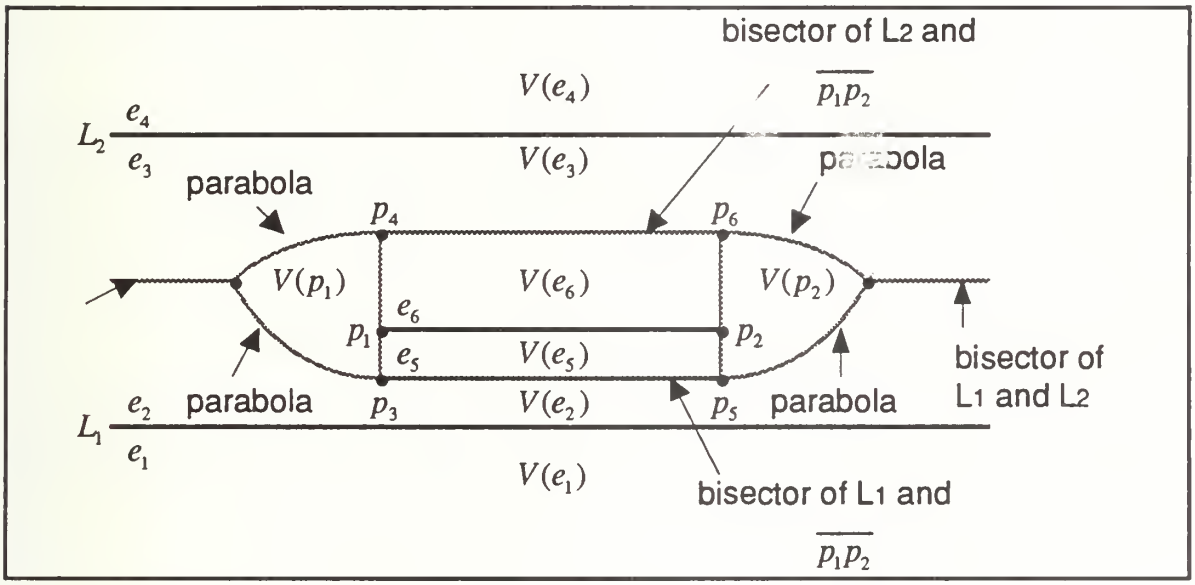

Figure 3.10 : Voronoi Diagram of Two Lines and A Line Segment

J. VORONOI DIAGRAM OF A NORMAL POLYGON

In the case that a world has only one *normal polygon*, we treat it as a union of vertices (points) p_i and open edges e_i. There are Voronoi boundaries to the polygon shown in (Figure 3.11).

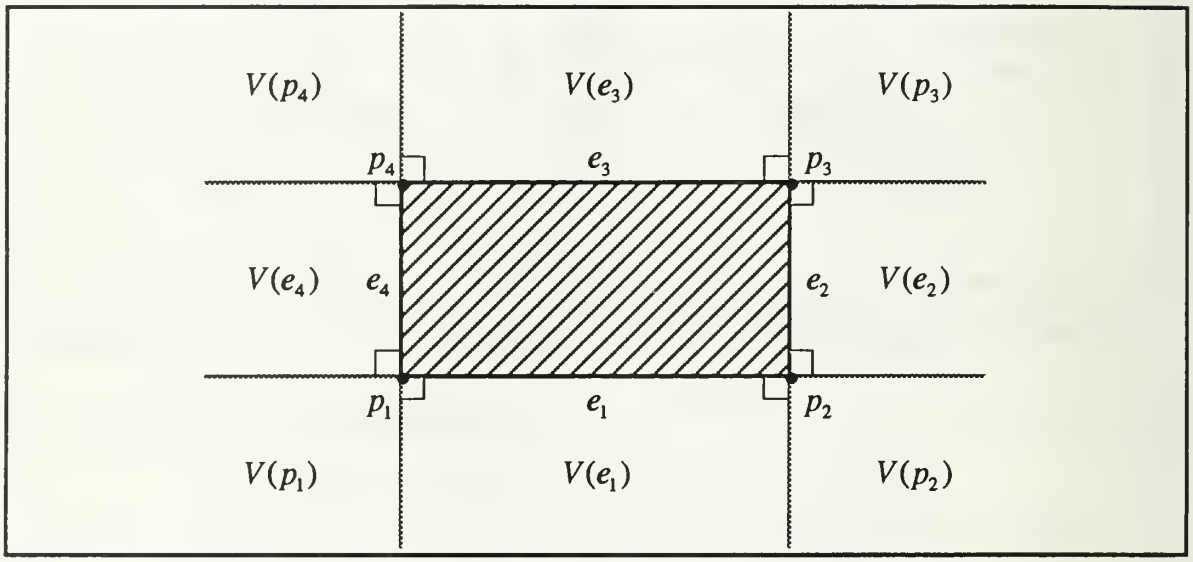

Figure 3.11 : Voronoi Diagram of A Normal Polygon

K. VORONOI DIAGRAM OF AN INVERTED POLYGON

In the case that a world has only one *inverted polygon*, we also treat it as a union of vertices (points) p_i and open edges e_i. There are Voronoi boundaries in its interior (Figure 3.12).

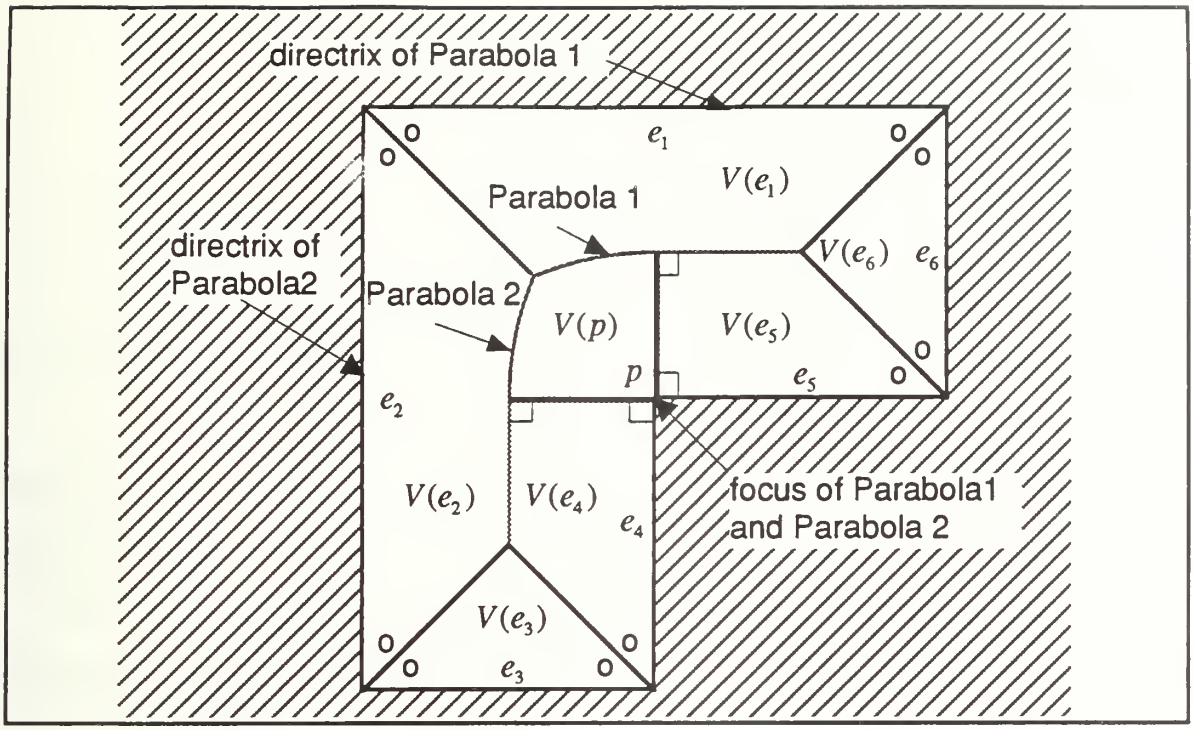

Figure 3.12 : Voronoi Diagram of An Inverted Polygon

IV. CURVE GENERATION

A. CONFIGURATION

To navigate a rigid body robot vehicle, the vehicle's state can be described by its current *configuration*,

$$q = (p, \theta) \tag{4.1}$$

where p is the vehicle's current coordinate position (x,y), and θ is the vehicle's tangent orientation at that point.

Let κ be the derivative of tangent orientation with respect to the length along the trajectory s which is called curvature.

$$\kappa(s) = \frac{d\theta(s)}{ds} \tag{4.2}$$

In this case, κ is not included in the configuration. However, κ can be included in the configuration if κ is required for the calculation.

B. NEXT FUNCTION

In order to compute the sequence of configurations, it is suffice to compute the next configuration at each step Δs. Given Δs and $\left(\frac{d\kappa}{ds}\right)$, we can estimate the vehicle's next configuration at $s + \Delta s$ as follows.

1. Short Circular Segment

The short path segment between s and $s + \Delta s$ is approximated by a circular curve segment (Figure 4.1).

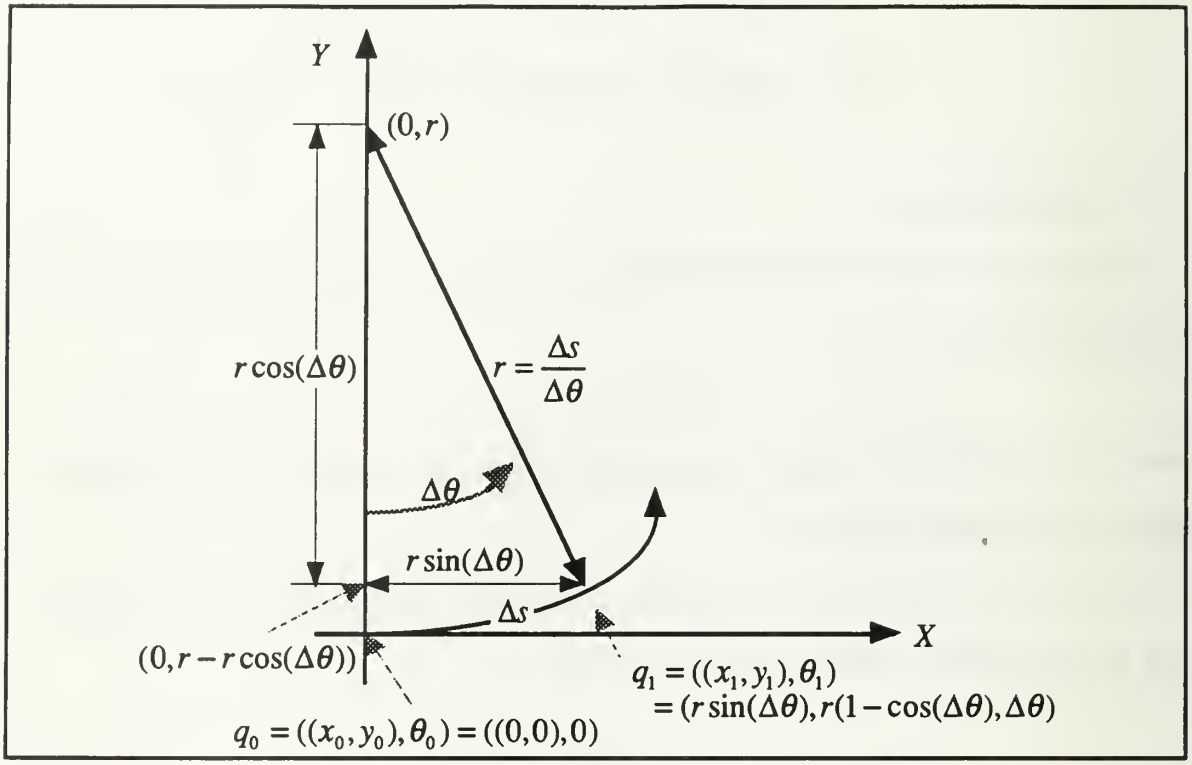

Figure 4.1 : Short Circular Segment

Assume configurations of both end points are q_0 and q_1.

$$q_0 = ((x_0, y_0), \theta_0) = ((0,0),0) \tag{4.3}$$

$$q_1 = ((x_1, y_1), \theta_1) \tag{4.4}$$

Let us assume $\Delta\theta \neq 0$. The radius of the short circular segment is

$$r = \frac{\Delta s}{\Delta\theta} \tag{4.5}$$

Where

$$\Delta\theta = \theta_1 - \theta_0 \tag{4.6}$$

The configuration of the end point q_1 is

$$q_1 = \begin{pmatrix} \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} \\ \theta_1 \end{pmatrix} = \begin{pmatrix} \begin{pmatrix} r\sin(\Delta\theta) \\ r(1-\cos(\Delta\theta)) \end{pmatrix} \\ \Delta\theta \end{pmatrix} = \begin{pmatrix} \begin{pmatrix} (\sin(\Delta\theta)/\Delta\theta)\Delta s \\ ((1-\cos(\Delta\theta))/\Delta\theta)\Delta s \end{pmatrix} \\ \Delta\theta \end{pmatrix} \text{ (if } \Delta\theta \neq 0) \quad (4.7)$$

$$q_1 = \begin{pmatrix} \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} \\ \theta_1 \end{pmatrix} = \begin{pmatrix} \begin{pmatrix} 0 \\ 0 \end{pmatrix} \\ 0 \end{pmatrix} \text{ (if } \Delta\theta = 0) \quad (4.8)$$

Using the Taylor expansion forms for the sin and cos functions

$$\frac{\sin(\Delta\theta)}{\Delta\theta} = \frac{\Delta\theta - (\Delta\theta)^3/3! + (\Delta\theta)^5/5! + }{\Delta\theta} = 1 - \frac{(\Delta\theta)^2}{3!} + \frac{(\Delta\theta)^4}{5!} - \cdots \quad (4.9)$$

$$\frac{1-\cos(\Delta\theta)}{\Delta\theta} = \frac{1-(1-(\Delta\theta)^2/2! + (\Delta\theta)^4/4! - (\Delta\theta)^6/6! \cdots)}{\Delta\theta}$$

$$= \left(\frac{1}{2!} - \frac{(\Delta\theta)^2}{4!} + \frac{(\Delta\theta)^4}{6!} - \cdots\right)\Delta\theta \quad (4.10)$$

Therefore, the configuration of the point q_1 is

$$q_1 = \begin{pmatrix} \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} \\ \theta_1 \end{pmatrix} = \begin{pmatrix} \begin{pmatrix} (1-(\Delta\theta)^2/3! + (\Delta\theta)^4/5! - \cdots)\Delta s \\ (1/2! - (\Delta\theta)^2/4! + (\Delta\theta)^4/6! - \cdots)\Delta\theta\Delta s \end{pmatrix} \\ \Delta\theta \end{pmatrix} \quad (4.11)$$

In general,

$$\Delta\theta \ll 1 \quad (4.12)$$

Therefore equation (4.11) can be approximated as

$$q_1 = \begin{pmatrix} \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} \\ \theta_1 \end{pmatrix} = \begin{pmatrix} \begin{pmatrix} (1-(\Delta\theta)^2/3!)\Delta s \\ (1/2! - (\Delta\theta)^2/4!)\Delta\theta\Delta s \end{pmatrix} \\ \Delta\theta \end{pmatrix} \quad (4.13)$$

2. Global Position Calculation

By using the composition function o [Ref. 1] which is a two dimensional coordinate transformation from the local coordinate system (x_1, y_1) to the global coordinate system (x_0, y_0), we can calculate the global position of the vehicle at s+Δs as follows.

$$q(s+\Delta s) = q_0 \circ q_1 \equiv \begin{pmatrix} x_0 + x_1 \cos\theta_0 - y_1 \sin\theta_0 \\ y_0 + x_1 \sin\theta_0 + y_1 \cos\theta_0 \\ \theta_1 + \theta_2 \end{pmatrix} \tag{4.14}$$

Where q_0 and q_1 are given Equation (4.3) and Equation (4.13).

V. SAFE NAVIGATION USING A STEERING FUNCTION

We use the derivative of curvature as the only control variable for the vehicle.

$$\frac{d\kappa}{ds} = f(p, \theta, \kappa) \tag{5.1}$$

Therefore the vehicle's motion is controlled only through changing its curvature.

$$\Delta\kappa = \left(\frac{d\kappa}{ds}\right)\Delta s \tag{5.2}$$

We propose the *steering function* in the following form

$$\frac{d\kappa}{ds} = -(a\Delta\kappa + b\Delta\theta + c\Delta d) \tag{5.3}$$

or

$$\frac{d\kappa}{ds} + a\Delta\kappa + b\Delta\theta + c\Delta d = 0 \tag{5.4}$$

where a, b, c are positive constants and $\Delta\kappa$, $\Delta\theta$, Δd are variables. Each evaluation of $\Delta\kappa$, $\Delta\theta$, Δd differs in the situation of the obstacles (in the case that the obstacles are one point and one line, the obstacles are two lines, and the obstacles are two points).

A. LINE TRACKING

Consider a special case of the line tracking (Figure 5.1). Assume we want to track the X-axis of the global coordinate system, then we can define three positive constants a, b, c .

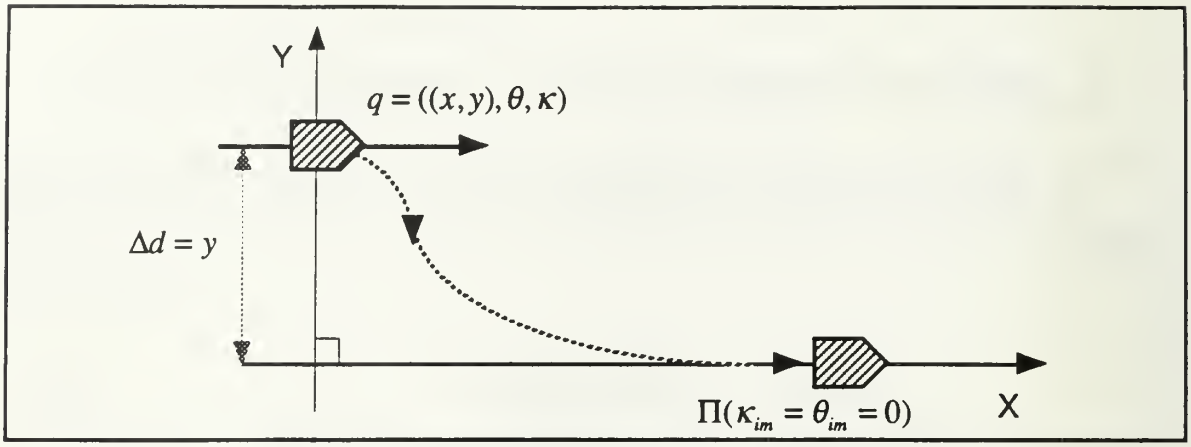

Figure 5.1 : Line Tracking

On the X-axis,

$$\kappa_{im} = \theta_{im} = 0 \tag{5.5}$$

Therefore

$$\Delta\kappa = \kappa - \kappa_{im} = \kappa \tag{5.6}$$

$$\Delta\theta = \theta - \theta_{im} = \theta \tag{5.7}$$

$$\Delta d = y \tag{5.8}$$

Let us now consider a short path segment (Figure 5.2).

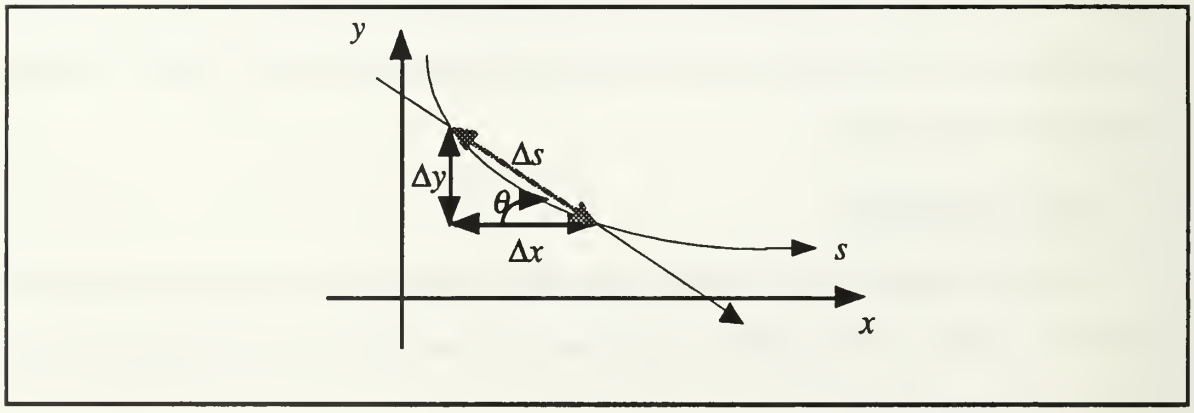

Figure 5.2 : Short Path Segment

The vehicle's tangent orientation θ is specified by

$$\tan\theta = \frac{\Delta y}{\Delta x} \tag{5.9}$$

Then

$$\begin{aligned}\theta &= \tan^{-1}\frac{\Delta y}{\Delta x} = \tan^{-1} y' \ (\text{as } \Delta x \to 0) \\ &= y' - \frac{(y')^3}{3} + \frac{(y')^5}{5} - \cdots\end{aligned} \tag{5.10}$$

Δs is defined as

$$\Delta s = \sqrt{(\Delta x)^2 + (\Delta y)^2} \tag{5.11}$$

Therefore

$$\frac{\Delta s}{\Delta x} = \frac{\sqrt{(\Delta x)^2 + (\Delta y)^2}}{\Delta x} = \sqrt{1 + \left(\frac{\Delta y}{\Delta x}\right)^2} = \sqrt{1 + (y')^2} \ (\text{as } \Delta x \to 0) \tag{5.12}$$

From the definition

$$\frac{d}{dx}(\tan^{-1} y') = \frac{\frac{d}{dx} y'}{1 + (y')^2} = \frac{y''}{1 + (y')^2} \tag{5.13}$$

From Equation (5.12) and Equation (5.13)

$$\kappa = \frac{d\theta}{ds} = \frac{d}{ds}(\tan^{-1} y') = \frac{\frac{d}{dx}(\tan^{-1} y')}{\frac{ds}{dx}} = \frac{\frac{y''}{1 + (y')^2}}{\sqrt{1 + (y')^2}} = \frac{y''}{(1 + (y')^2)^{\frac{3}{2}}} \tag{5.14}$$

Therefore

$$\frac{d\kappa}{ds} = \frac{\frac{d\kappa}{dx}}{\frac{ds}{dx}} = \frac{\frac{y''}{(1 + (y')^2)^{\frac{3}{2}}}}{\sqrt{1 + (y')^2}} = y'''(1 + (y')^2)^{-2} - 3y'(y'')^2(1 + (y')^2)^{-3} \tag{5.15}$$

Assume

$$y'^2 \ll 1 \tag{5.16}$$

$$y'(y'')^2 \ll y''' \tag{5.17}$$

Then from Equation (5.10)

$$\Delta\theta = y' \tag{5.18}$$

From Equation (5.14)

$$\Delta\kappa = y'' \tag{5.19}$$

From Equation (5.15)

$$\frac{d\kappa}{ds} = y''' \tag{5.20}$$

Equation (5.4) becomes an ordinary differential equation:

$$y''' + ay'' + by' + cy = 0 \tag{5.21}$$

$$(D^3 + aD^2 + bD + c)y = 0 \tag{5.22}$$

Since this is a third order linear homogeneous ordinary differential equation with constant coefficients, it must have at least one real root. If it has a non-negative root, it does not have a converging solution. If it has a complex conjugate root, the solution oscillates even if it decays. Since we want non-oscillatory decaying solutions, Equation (5.22) must have three negative roots of D. Also if we want a critical damping solution then Equation (5.22) must be specified to have a triple root, $-k$ (where $k > 0$).

$$D^3 + aD^2 + bD + c \equiv (D+k)^3 = D^3 + 3kD^2 + 3k^2D + k^3 \tag{5.23}$$

Therefore if we choose

$$a = 3k \tag{5.24}$$

$$b = 3k^2 \tag{5.25}$$

$$c = k^3 \tag{5.26}$$

Equation (5.22) becomes

$$(D+k)^3 y = 0 \tag{5.27}$$

Thus, under this condition, there is only one degree of freedom in choosing the parameter k instead of three parameters a, b and c. We define

$$s_0 \equiv \frac{1}{k} \tag{5.28}$$

This size constant s_0 controls the distance for which the vehicle runs before it gets on track. A smaller size constant makes the transition distance smaller. Thus s_0 controls the sharpness of the trajectory. From Equations (5.3), (5.24), (5.25), (5.26) and (5.28), the revised steering function becomes

$$\frac{d\kappa}{ds} = -\left(3\left(\frac{1}{s_0}\right)\Delta\kappa + 3\left(\frac{1}{s_0}\right)^2 \Delta\theta + 3\left(\frac{1}{s_0}\right)^3 \Delta d\right) \tag{5.29}$$

$\Delta\kappa$, $\Delta\theta$, Δd are evaluated depending upon the environment. Let consider three situations for vehicle navigation: the objects are two points, the objects are two lines, the objects are one point and one line.

B. TWO POINTS

When we navigate the vehicle safely to make the same clearance from two points, its trajectory becomes a line which is a bisector of the two points. It is a Voronoi boundary of the two points. Let consider the case of the world consists of two points p_1 and p_2 (Figure 5.3).

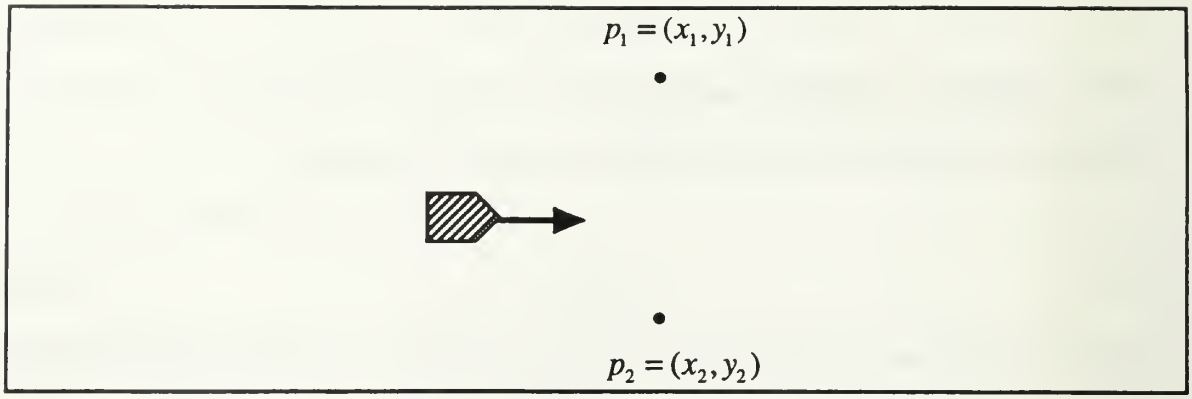

Figure 5.3 : Safe Navigation of Two Points

1. **Evaluation of $\Delta\kappa$**

When the vehicle's configuration is $q = (p, \theta, \kappa)$,

$$\Delta\kappa = \kappa \tag{5.30}$$

Since final value of κ on the Voronoi boundary is zero.

2. **Evaluation of $\Delta\theta$**

Let $\Psi(p, p_1)$ denote the orientation from p to p_1 and $\Psi(p, p_2)$ denote the orientation from p to p_2. Let α be the difference between the orientation $\Psi(p, p_1)$ and the vehicle's orientation θ; β is the difference between the vehicle's orientation θ and $\Psi(p, p_2)$ (Figure 5.4).

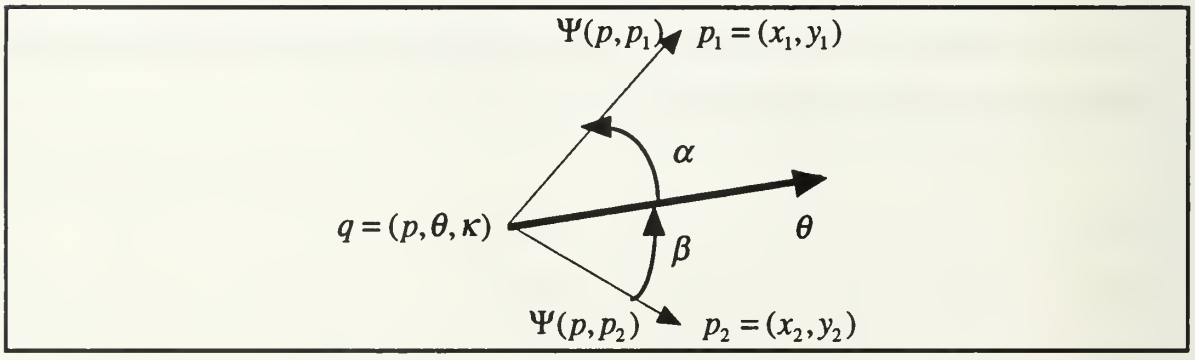

Figure 5.4 : Evaluation of $\Delta\theta$

$$\alpha = \Psi(p, p_1) - \theta \tag{5.31}$$

$$\beta = \theta - \Psi(p, p_2) \tag{5.32}$$

When vehicle is on the Voronoi boundary,

$$\alpha = \beta \tag{5.33}$$

Let the desired orientation be θ_d. At the start point

$$\theta_d = normalize1\left(\frac{\Psi(p, p_1) + \Psi(p, p_2)}{2} - \Psi(p, p_0)\right) + \Psi(p, p_0) \tag{5.34}$$

Where normalize1 is a function which normalizes its argument into a range of $\left[-\frac{\pi}{2}, \frac{\pi}{2}\right]$ by addition of $\pm n\pi$ if necessary and p_0 is the middle point between p_1 and p_2. At the other point,

$$\theta_d = normalize1\left(\frac{\Psi(p, p_1) + \Psi(p, p_2)}{2} - \theta_{prev}\right) + \theta_{prev} \tag{5.35}$$

Where θ_{prev} is the vehicle's previous desired orientation θ_d at the point. Then the variable $\Delta\theta$ is evaluated as

$$\Delta\theta = \theta - \theta_d \tag{5.36}$$

3. Evaluation of Δd

Let d_1 be the distance between p and p_1, and d_2 be the distance from p to p_2.

$$d_1 = \sqrt{(x - x_1)^2 + (y - y_1)^2} \tag{5.37}$$

$$d_2 = \sqrt{(x - x_2)^2 + (y - y_2)^2} \tag{5.38}$$

The signed variable Δd is evaluated as follows (Figures 5.5 and 5.6). Note that Δd can be signed positive or negative, or equal zero.

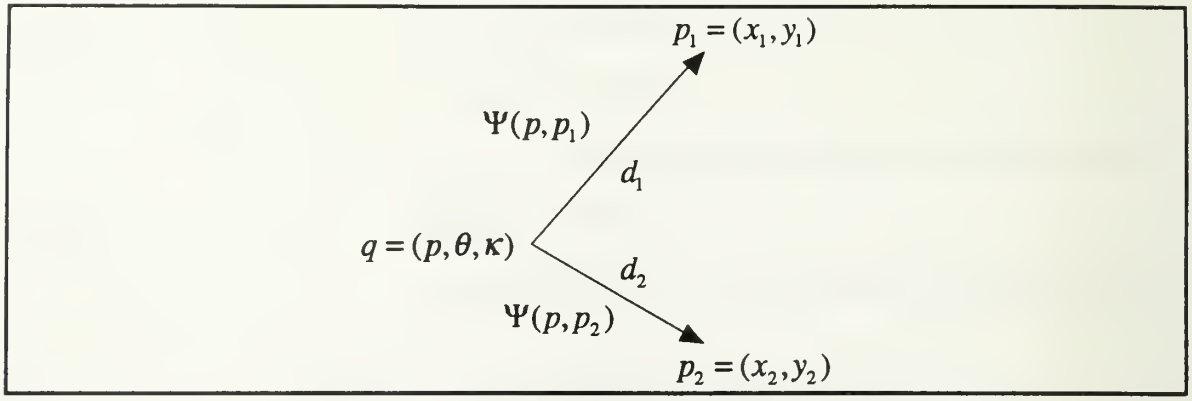

Figure 5.5 : Case $\Psi(\mathbf{p},\mathbf{p}_1) - \Psi(\mathbf{p},\mathbf{p}_2) > 0$

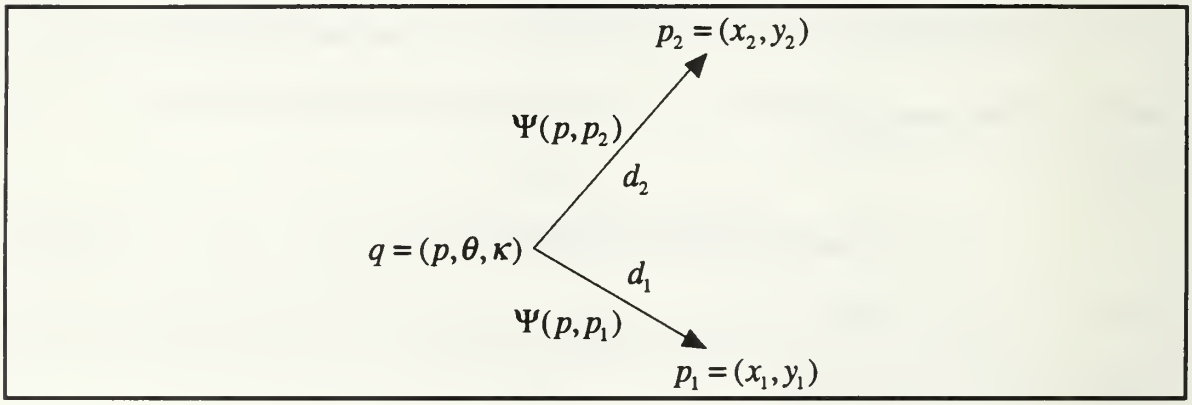

Figure 5.6 : Case $\Psi(\mathbf{p},\mathbf{p}_1) - \Psi(\mathbf{p},\mathbf{p}_2) < 0$

$$\Delta d = d_2 - d_1 \quad (\text{if } \Psi(p, p_1) - \Psi(p, p_2) > 0) \tag{5.39}$$

$$\Delta d = d_1 - d_2 \quad (\text{if } \Psi(p, p_1) - \Psi(p, p_2) < 0) \tag{5.40}$$

4. Simulation Results

The use of this steering function for vehicle motion control is demonstrated by algorithmic simulation and by use on the existing robot Yamabico 11. The result is shown in Figures 5.7 and 5.8. Figure 5.7 shows the case $p_1 = (0,100)$, $p_2 = (0,-100)$, the initial configuration of the vehicle is $q = ((-300,50),\theta,0)$, where there are eight cases of the initial θ : 0, 45, 90, 135,

180, 225, 270, 315 degrees. Figure 5.8 shows the case where the initial configuration of the vehicle is $q = ((-300,-50),\theta,0)$.

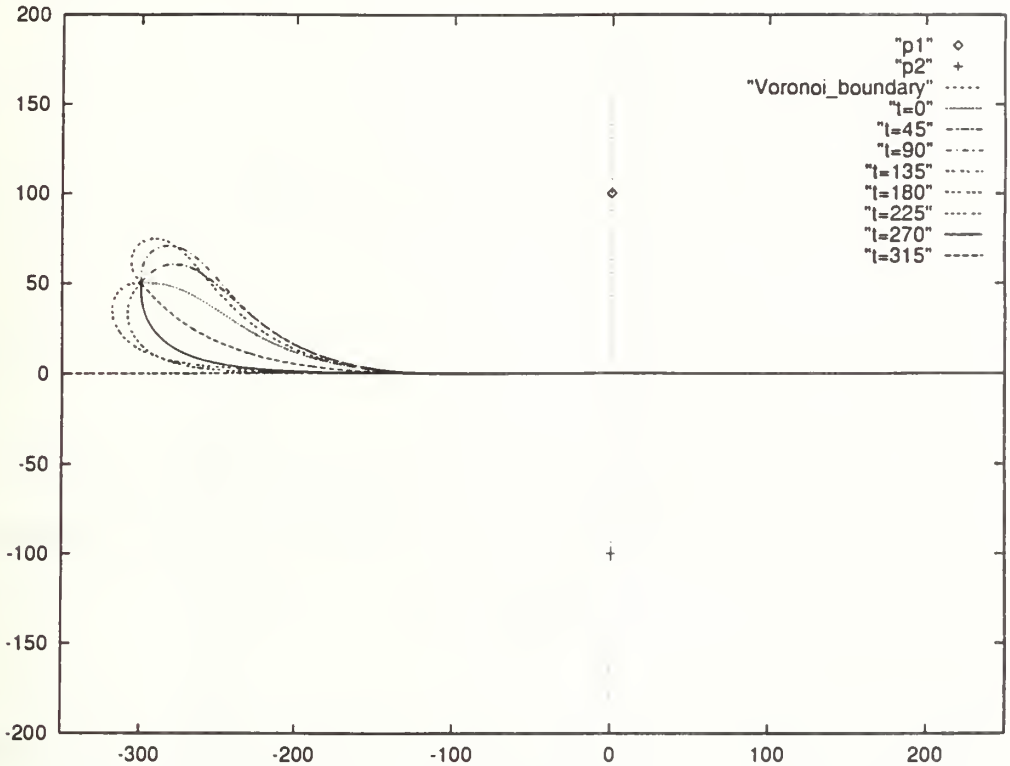

Figure 5.7 : Simulation Result of Two Points, q = ((–300,50),θ,0)

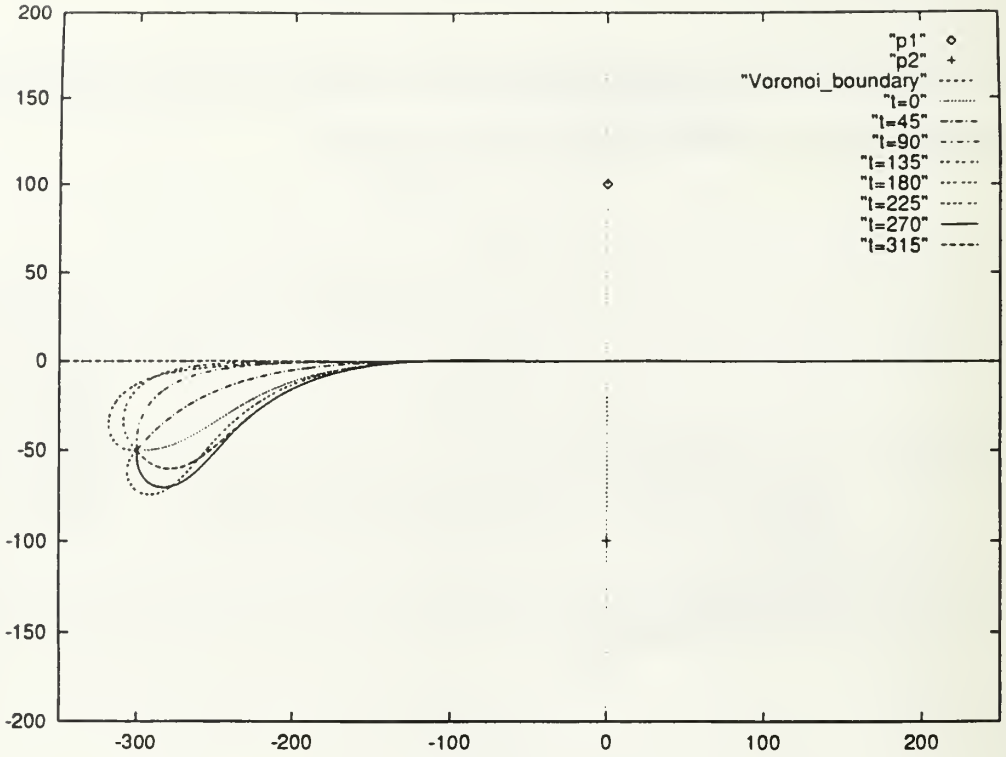

Figure 5.8 : Simulation Result of Two Points, $q = ((-300,-50),\theta,0)$

C. TWO LINES

When we navigate the vehicle safely to make the same clearance from two lines, its trajectory becomes a line which is a bisector of two directed lines. So it is a Voronoi boundary of two lines. Assume a world consists of two directed lines q_1 and q_2, there are two cases: they are parallel or not parallel (Figures 5.9 and 5.10). Interestingly, calculations for $\Delta\kappa$, $\Delta\theta$ and Δd are identical in each case.

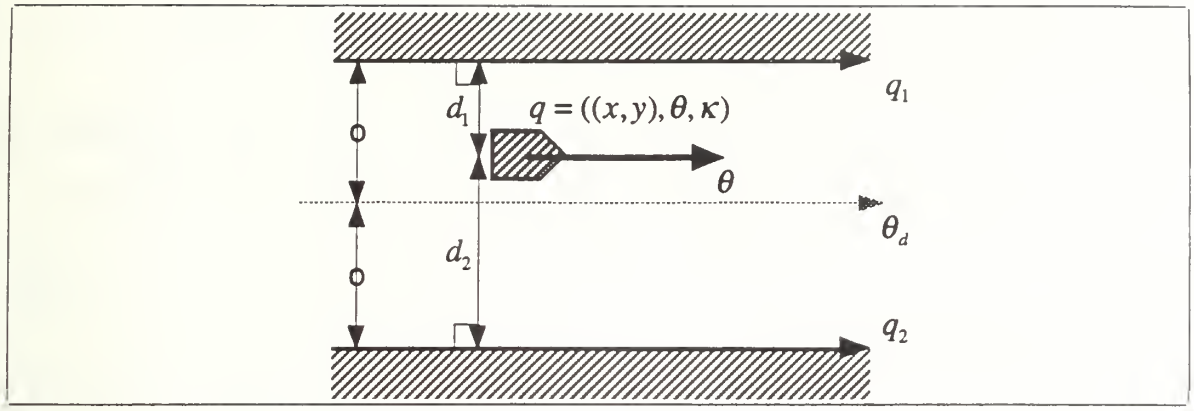

Figure 5.9 : Parallel Directed Lines

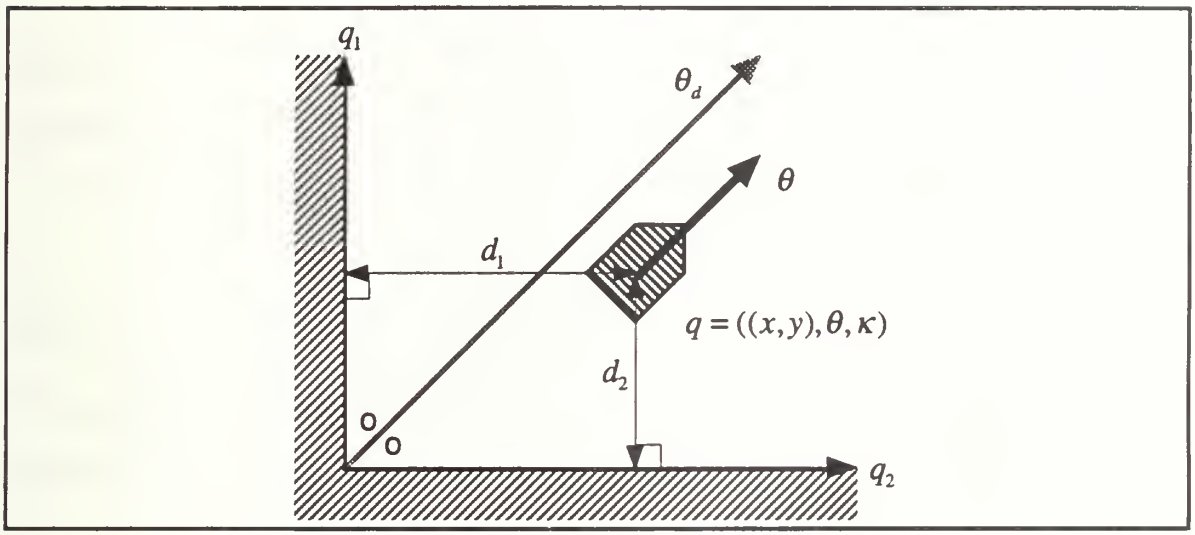

Figure 5.10 : Not Parallel Directed Lines

1. Evaluation of $\Delta\kappa$

When the vehicle's configuration is $q = (p,\theta,\kappa)$,

$$\Delta\kappa = \kappa \tag{5.41}$$

Since final value of κ on Voronoi boundary is zero.

2. Evaluation of $\Delta\theta$

When the vehicle is on the Voronoi boundary its orientation is the average of θ_1 and θ_2. Let this desired orientation be θ_d,

$$\theta_d = normalize1\left(\frac{\theta_1 + \theta_2}{2} - \theta_1\right) + \theta_1 \tag{5.42}$$

Where normalize1 is a function which normalizes its argument into a range of $\left[-\frac{\pi}{2}, \frac{\pi}{2}\right]$ by addition of $\pm n\pi$ if necessary. Then the variable $\Delta\theta$ is evaluated as

$$\Delta\theta = \theta - \theta_d \tag{5.43}$$

3. Evaluation of Δd

Let d_1 be the signed distance from p to q_1, and d_2 be the signed distance from p to q_2.

$$d_1 = -(x - x_1)\sin\theta_1 + (y - y_1)\cos\theta_1 \tag{5.44}$$

$$d_2 = -(x - x_2)\sin\theta_2 + (y - y_2)\cos\theta_2 \tag{5.45}$$

The signed variable Δd is evaluated as

$$\Delta d = \frac{d_1 + d_2}{2} \tag{5.46}$$

The signed distance from p to q_1 is the distance between p and q_1. If p is on the left of q_1, then $d_1 > 0$ and if p is on the right of q_1, then $d_1 < 0$ (Figure 5.11). A similar argument holds for d_2.

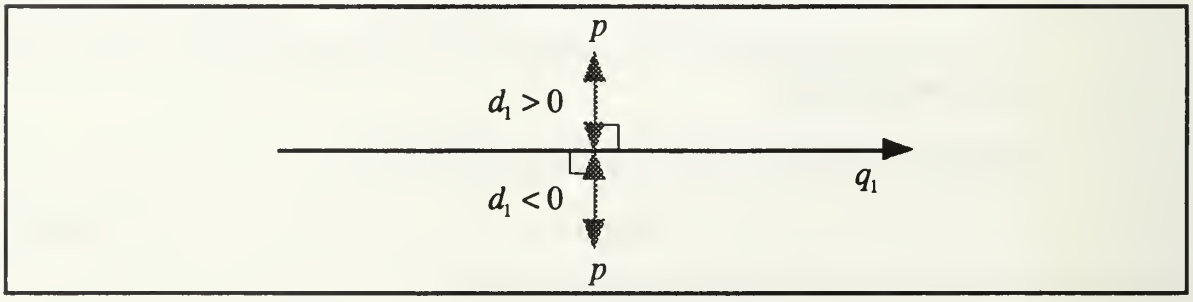

Figure 5.11 : Signed Distance from p to q_1

4. Simulation Result

The result of algorithmic simulation can be found in Figures 5.12, 5.13, 5.14, and 5.15.

Figures 5.12 and 5.13 show the case where the lines are parallel. Figure 5.12 shows the case $q_1 = ((0,100),0,0)$, $q_2 = ((0,-100),0,0)$ and the initial configuration of the vehicle is $q = ((0,50),\theta,0)$, where there are eight cases of the initial θ : 0, 45, 90, 135, 180, 225, 270, 315 degrees. Figure 5.13 shows the case where the initial configuration of the vehicle is $q = ((0,-50),\theta,0)$.

Figures 5.14 and 5.15 show the case where the lines are not parallel. Figure 5.14 shows the case $q_1 = ((0,0),90,0)$, $q_2 = ((0,0),0,0)$ and the initial configuration of the vehicle is $q = ((50,150),\theta,0)$. Figure 5.15 shows the case where the initial configuration of the vehicle is $q = ((150,50),\theta,0)$.

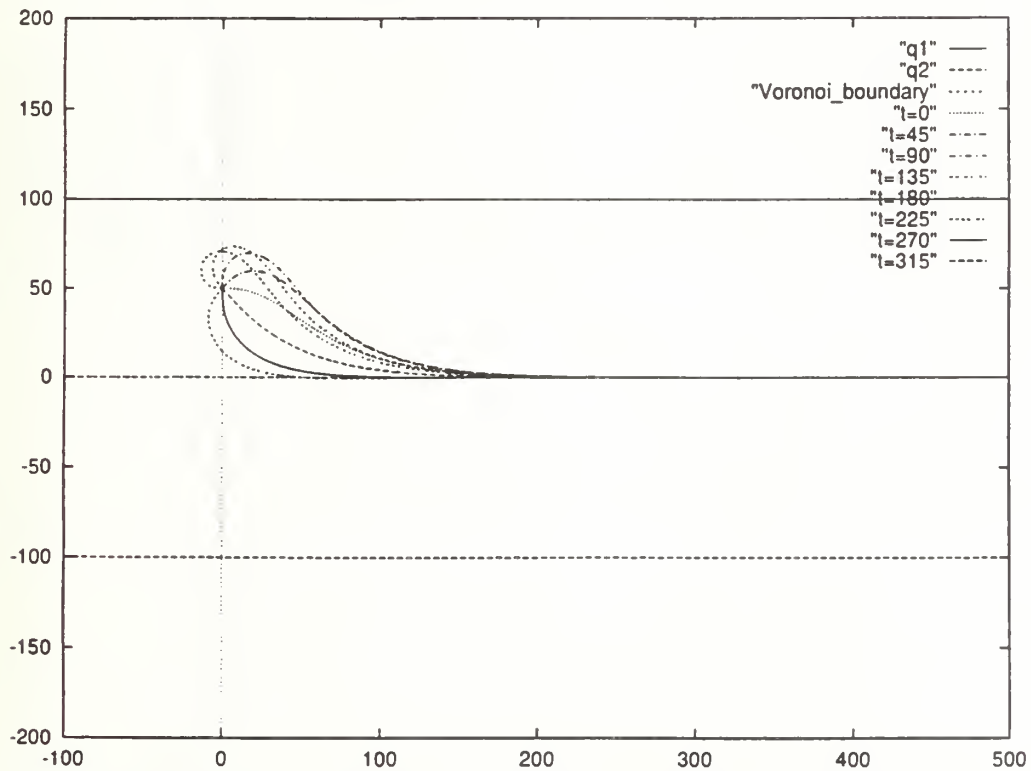

Figure 5.12 : Simulation Result of Parallel lines, $q = ((0,50),\theta,0)$

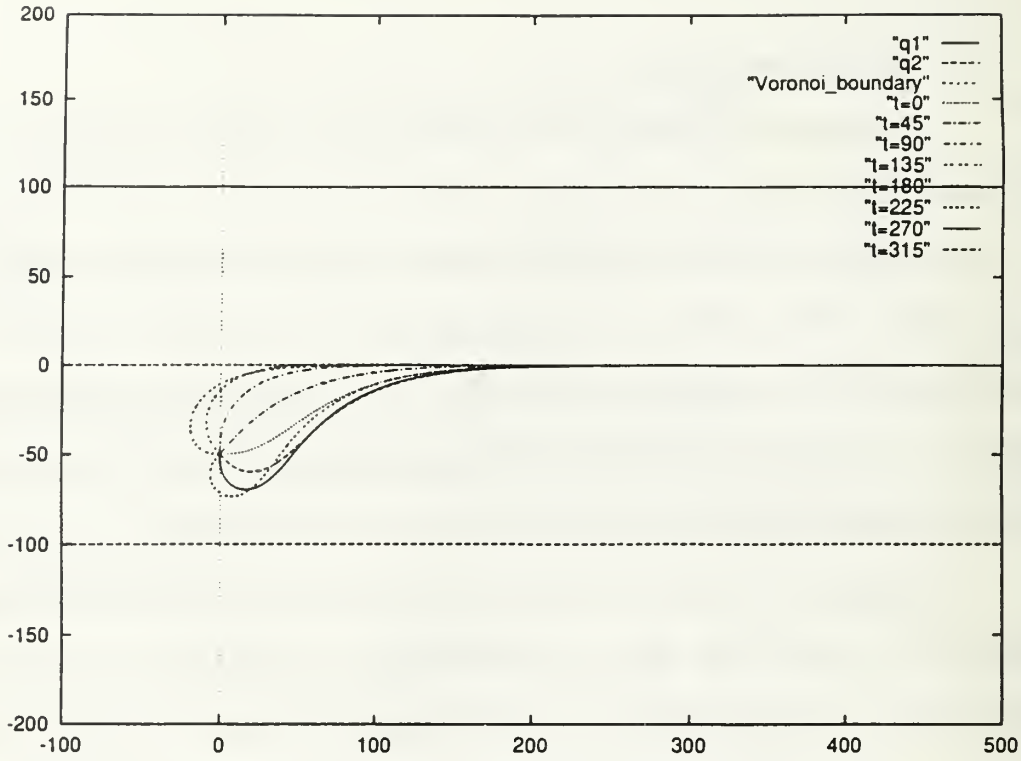

Figure 5.13 : Simulation Result of Parallel Lines, q = ((0,–50),θ,0)

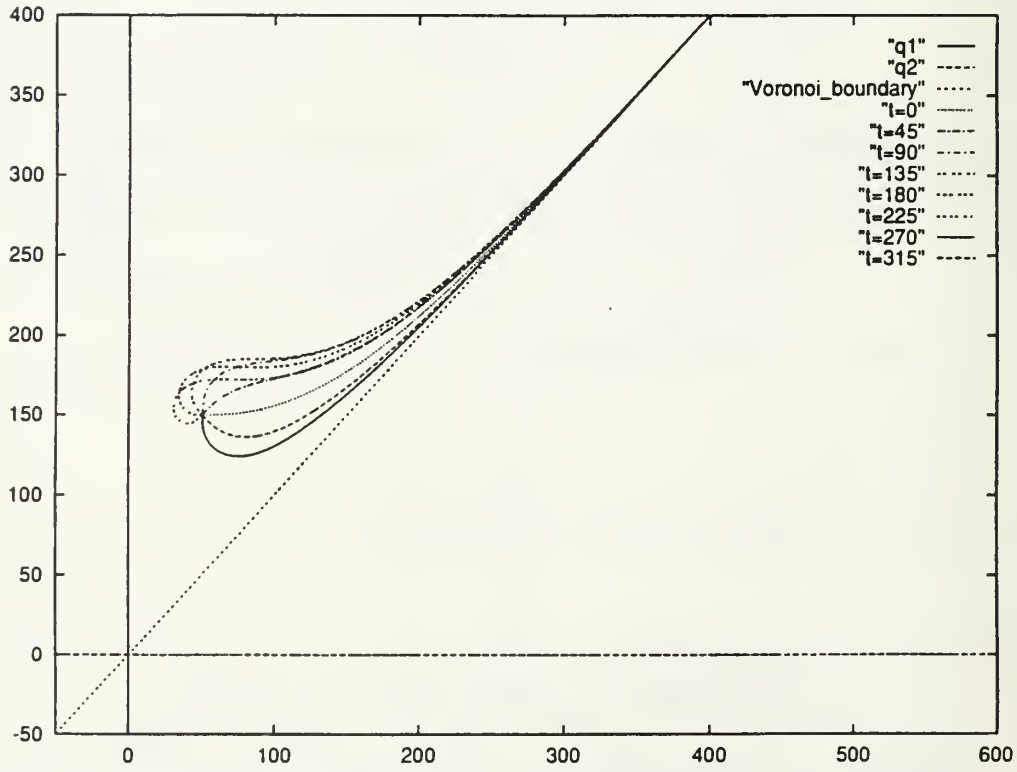

Figure 5.14 : Simulation Result of Not Parallel Lines, q = ((50,150),θ,0)

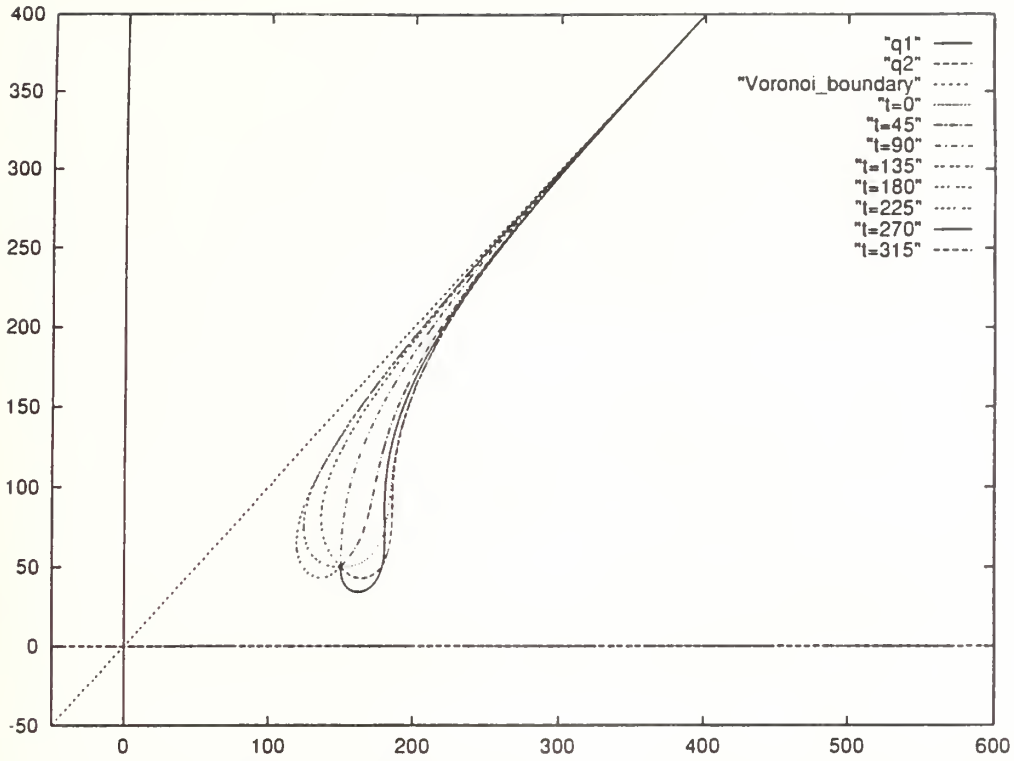

Figure 5.15 : Simulation Result of Not Parallel Lines, q = ((150,50),θ,0)

D. ONE POINT AND ONE LINE

When we navigate the vehicle safely to make the same clearance from one point and one line, its trajectory becomes a parabola. So it is a Voronoi boundary of a point and a line.

1. Definition of Parabola

When a world consists of a point p_f and a directed line q_0, its Voronoi boundary becomes a parabola. A parabola is defined as the focus p_f and the directrix q_0 :

$$p_f = (x_f, y_f) \tag{5.47}$$

$$q_0 = (x_0, y_0, \theta_0) \tag{5.48}$$

The directrix q_0 has a direction, and hence, parabola has a direction. Let l be the signed distance from p_f to q_0.

$$l = -(x_f - x_0)\sin\theta_0 + (y_f - y_0)\cos\theta_0 \qquad (5.49)$$

The signed distance from p_f to q_0 is the distance between p_f and q_0. If p_f is on the left of q_0, then $l > 0$ and if p_f is on the right of q_0, then $l < 0$ (Figure 5.16).

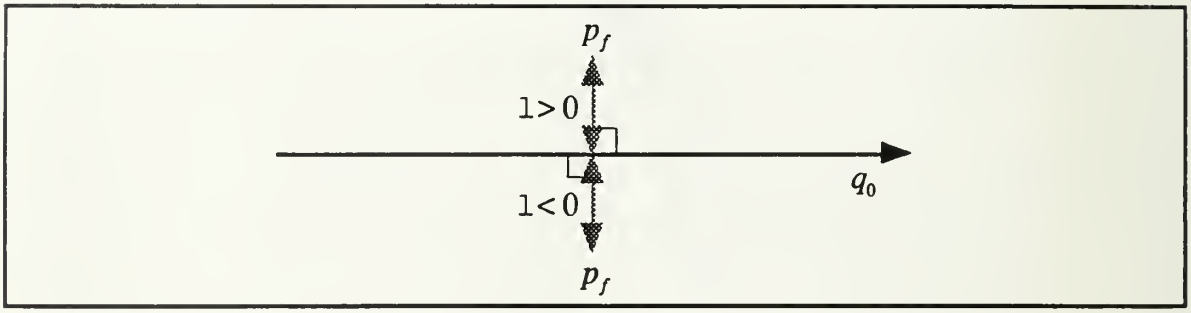

Figure 5.16 : Signed Distance from p_f to q_0

a. Case $l > 0$

Let θ_1 denote the orientation of the normal ray from p_f to q_0. We define a polar coordinate system whose pole is p_f and whose *initial ray* is θ_1 (Figure 5.17).

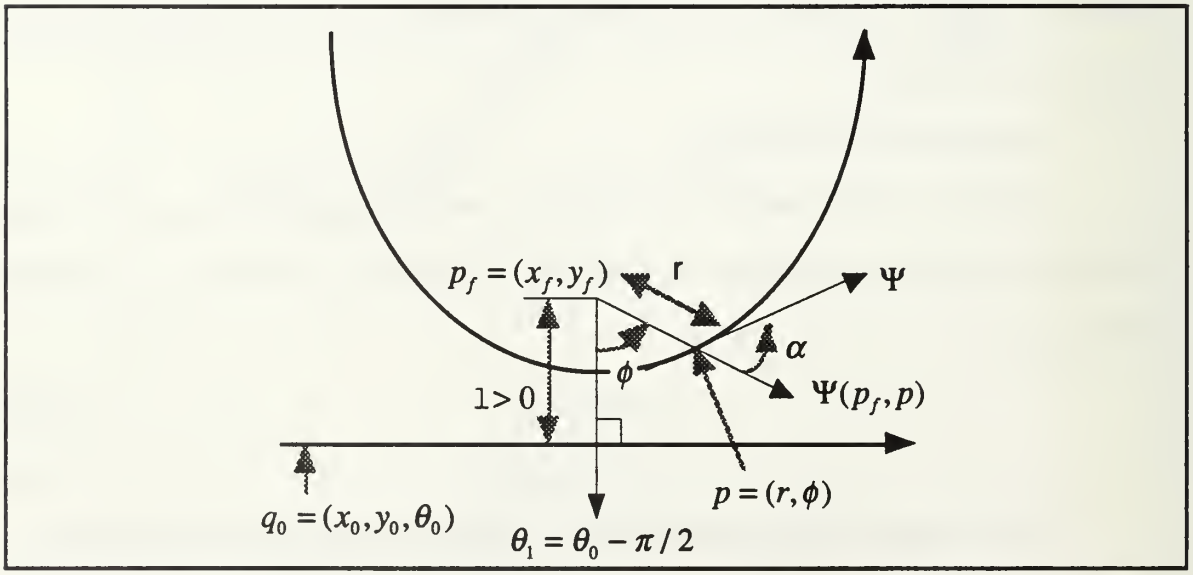

Figure 5.17 : Parabola ($l > 0$)

$$\theta_1 = \theta_0 - \pi/2 \tag{5.50}$$

In this system, p is represented by (r,ϕ), where r is the distance between p_f and p and ϕ is the orientation from p_f to p taken from the initial ray. In this case, we take ϕ $(-\pi < \phi < \pi)$ counterclockwise from the initial ray. The coordinate of p in the global Cartesian system is

$$\begin{aligned}(x,y) &= (x_f + r\cos(\theta_1+\phi), y_f + r\sin(\theta_1+\phi)) \\ &= \left(x_f + \frac{1\sin(\theta_0+\phi)}{1+\cos\phi}, y_f - \frac{1\cos(\theta_0+\phi)}{1+\cos\phi}\right)\end{aligned} \tag{5.51}$$

Let $\Psi(p_f,p)$ denote the orientation from p_f to p. By definition, the angle α between $\Psi(p_f,p)$ and the orientation Ψ of the tangent at p is defined as

$$\alpha = \frac{\pi}{2} - \frac{\phi}{2} \tag{5.52}$$

The orientation Ψ of the tangent at p in the polar coordinate system is

$$\Psi = \phi + \alpha = \phi + \left(\frac{\pi}{2} - \frac{\phi}{2}\right) = \frac{\pi}{2} + \frac{\phi}{2} \tag{5.53}$$

The orientation θ of this tangent at p in the global coordinate system is

$$\theta = \theta_1 + \Psi = \left(\theta_0 - \frac{\pi}{2}\right) + \left(\frac{\pi}{2} + \frac{\phi}{2}\right) = \frac{\phi}{2} + \theta_0 \tag{5.54}$$

Let s denote the arc length. Then the curvature κ at p is

$$\kappa = \frac{d\theta}{ds} = \frac{\frac{d\theta}{d\phi}}{\frac{ds}{d\phi}} = \frac{\frac{1}{2}}{\sqrt{(\frac{dr}{d\phi})^2 + r^2}} = \frac{1}{2\sqrt{\frac{l^2}{(1+\cos\phi)^2} + \frac{l^2}{(1+\cos\phi)^4}}}$$

$$= \frac{(1+\cos\phi)^2}{2l\sqrt{(1+\cos\phi)^2 + \sin^2\phi}} = \frac{(1+\cos\phi)^2}{2l\sqrt{2+2\cos\phi}} = \frac{1}{2\sqrt{2}l}(1+\cos\phi)^{\frac{3}{2}}$$

$$= \frac{1}{2\sqrt{2}l}\left(2\cos^2\left(\frac{\phi}{2}\right)\right)^{\frac{3}{2}} = \frac{1}{l}\cos^3\left(\frac{\phi}{2}\right) \qquad (5.55)$$

b. Case $l < 0$

In this case (Figure 5.18), the orientation θ_1 of the initial ray in the global coordinate system is

$$\theta_1 = \theta_0 + \pi/2 \qquad (5.56)$$

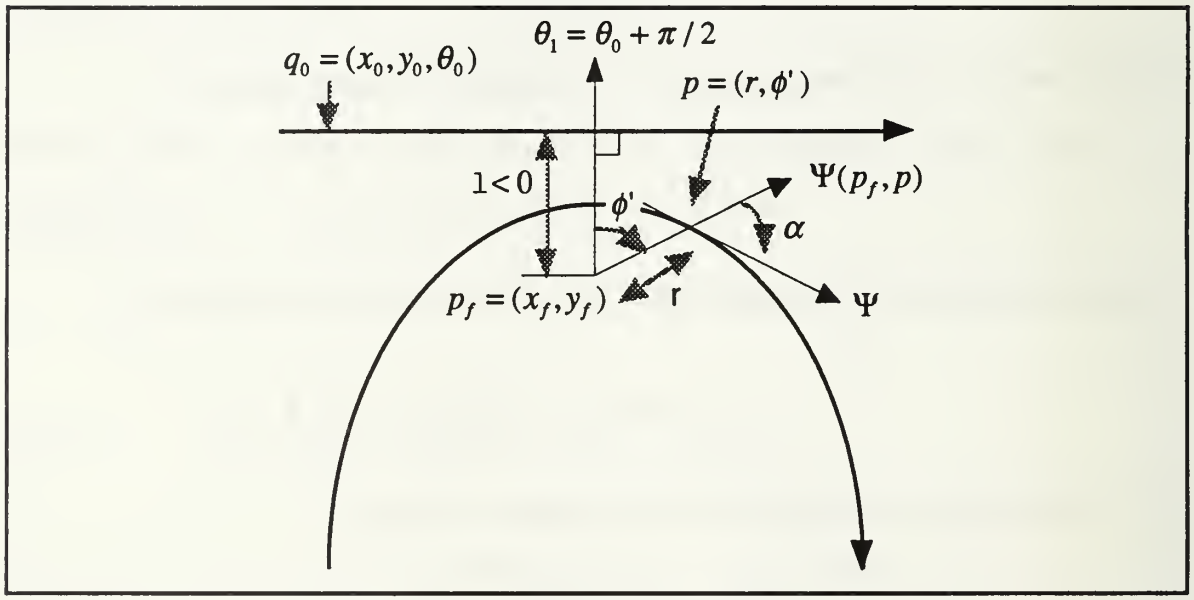

Figure 5.18 : Parabola ($l < 0$)

We take ϕ $(-\pi < \phi < \pi)$ clockwise from the initial ray.

$$\phi = -\phi' \qquad (5.57)$$

Then the coordinate of p in the global Cartesian system is defined by Equation

(5.51), the orientation θ of the tangent at p in the global coordinate system is defined by Equation (5.55) and the curvature κ at p is defined by Equation (5.56).

2. Evaluation of $\Delta\kappa$

When the vehicle's configuration is $q=(p,\theta,\kappa)$, assume the intersection of the orientation $\Psi(p,p_f)$ and the parabola is $q_{para}=(p_{para},\theta_{para},\kappa_{para})$ as shown in Figure 5.19. The variable $\Delta\kappa$ is evaluated as

$$\Delta\kappa=\kappa-\kappa_{para} \tag{5.58}$$

Note that $\Delta\kappa$ converges to zero as q approaches to q_{para}.

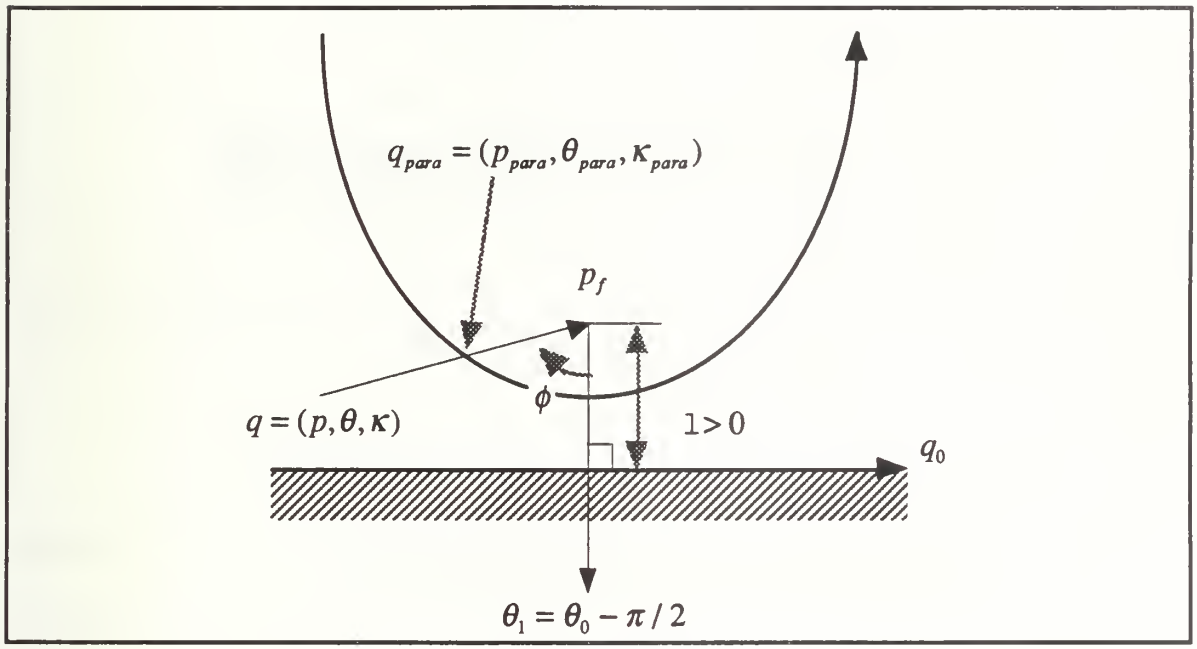

Figure 5.19 : Evaluation of $\Delta\kappa$

3. Evaluation of $\Delta\theta$

Let α be the difference between the vehicle's orientation θ and the orientation θ_1 of the initial ray. Also let β be the difference between the orientation $\Psi(p, p_f)$ and the vehicle's orientation θ (Figure 5.20).

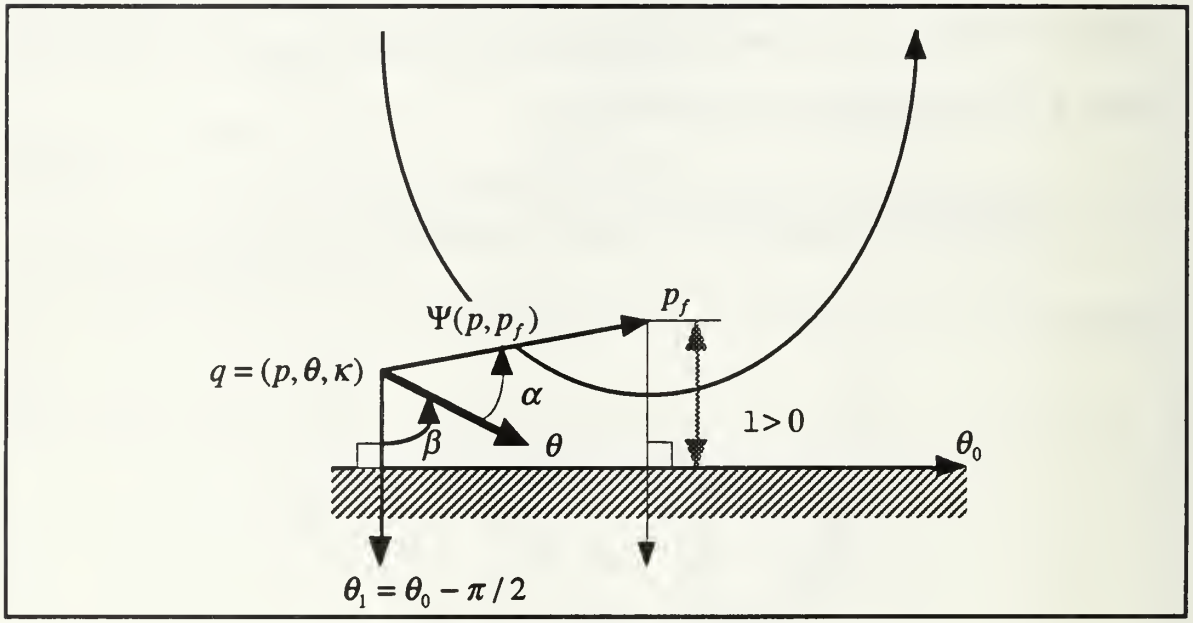

Figure 5.20 : Evaluation of $\Delta\theta$

Then

$$\alpha = \Psi(p, p_f) - \theta \tag{5.59}$$

$$\beta = \theta - \theta_1 \tag{5.60}$$

When vehicle is on the parabola,

$$\alpha = \beta \tag{5.61}$$

Let this desired orientation be θ_d,

$$\theta_d = normalize1\left(\frac{\Psi(p,p_f)-\theta_1}{2}-\theta_0\right)+\theta_0 \tag{5.62}$$

Where normalize1 is a function which normalizes its argument into a range of $\left[-\frac{\pi}{2},\frac{\pi}{2}\right]$ by addition of $\pm n\pi$ if necessary. Then the variable $\Delta\theta$ is evaluated as

$$\Delta\theta = \theta - \theta_d \tag{5.63}$$

4. Evaluation of Δd

Let d_1 be the distance between p and p_f, and d_2 be the signed distance from p to q_0 (Figure 5.21).

$$d_1 = \sqrt{(x-x_f)^2+(y-y_f)^2} \tag{5.64}$$

$$d_2 = -(x-x_0)\sin\theta_0 + (y-y_0)\cos\theta_0 \tag{5.65}$$

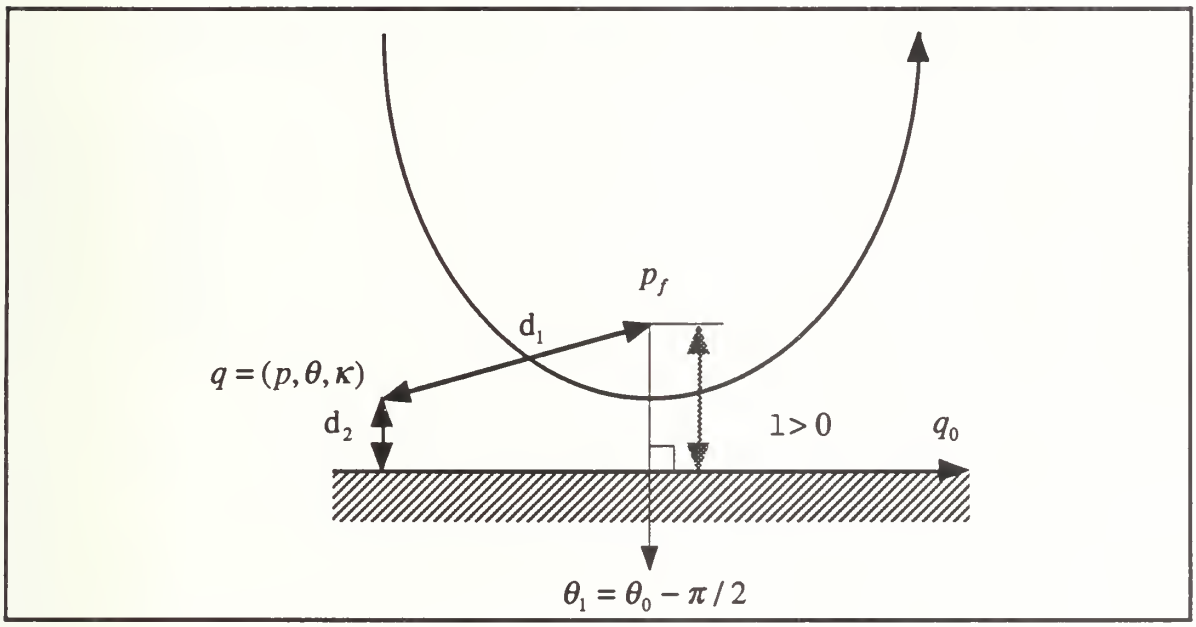

Figure 5.21 : Evaluation of Δd

The signed variable Δd is evaluated as

$$\Delta d = d_2 - d_1 \text{ (if } l>0) \tag{5.66}$$

$$\Delta d = d_2 + d_1 \text{ (if } l<0) \tag{5.67}$$

5. Simulation Result

The result of algorithmic simulation can be found in Figures 5.22, 5.23, 5.24 and 5.25. Figures 5.22 and 5.23 show the case where the l is positive. Figures 5.24 and 5.25 show the case where the l is negative.

Figure 5.22 shows the case of $p_f = (0,200)$, $q_0 = ((0,0),0,0)$, the initial configuration of the vehicle is $q = ((-300,200),\theta,0)$, where there are eight cases of the initial θ : 0, 45, 90, 135, 180, 225, 270, 315 degrees. Figure 5.23 shows the case where the initial configuration of the vehicle is $q = ((-200,300),\theta,0)$.

Figure 5.24 shows the case of $p_f = (0,-200)$, $q_0 = ((0,0),0,0)$, the initial configuration of the vehicle is $q = ((-300,-200),\theta,0)$, where there are eight cases of the initial θ : 0, 45, 90, 135, 180, 225, 270, 315 degrees. Figure 5.25 shows the case where the initial configuration of the vehicle is $q = ((-200,-300),\theta,0)$.

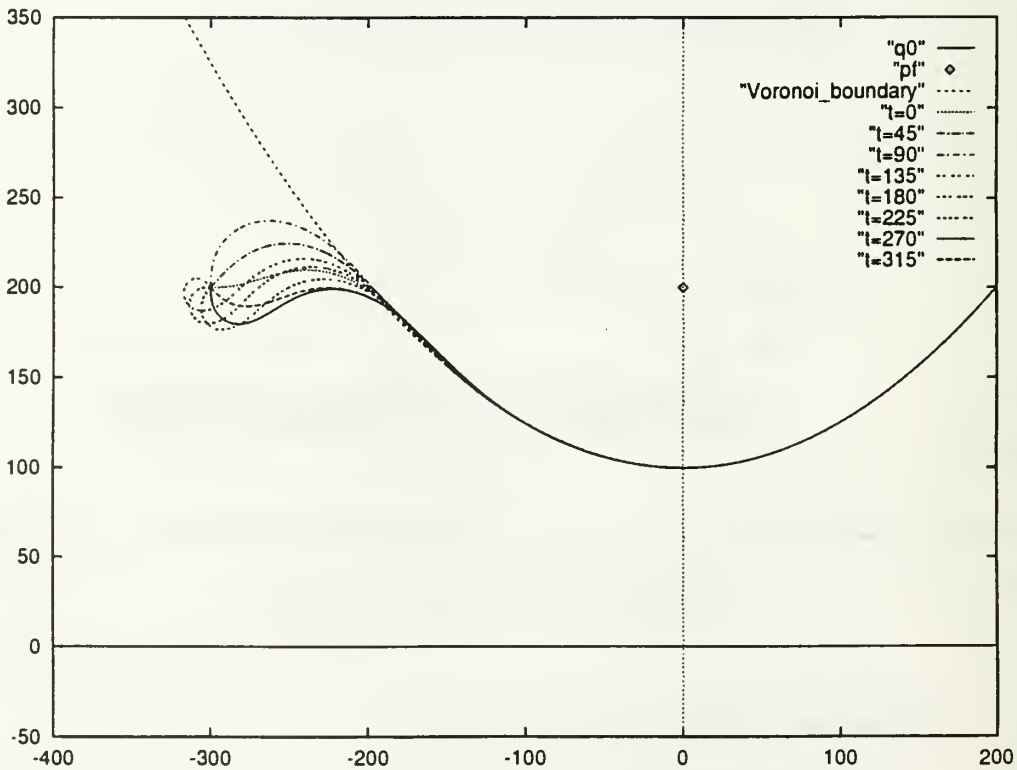

Figure 5.22 : Simulation Result of l> 0, q = ((−300,200),θ,0)

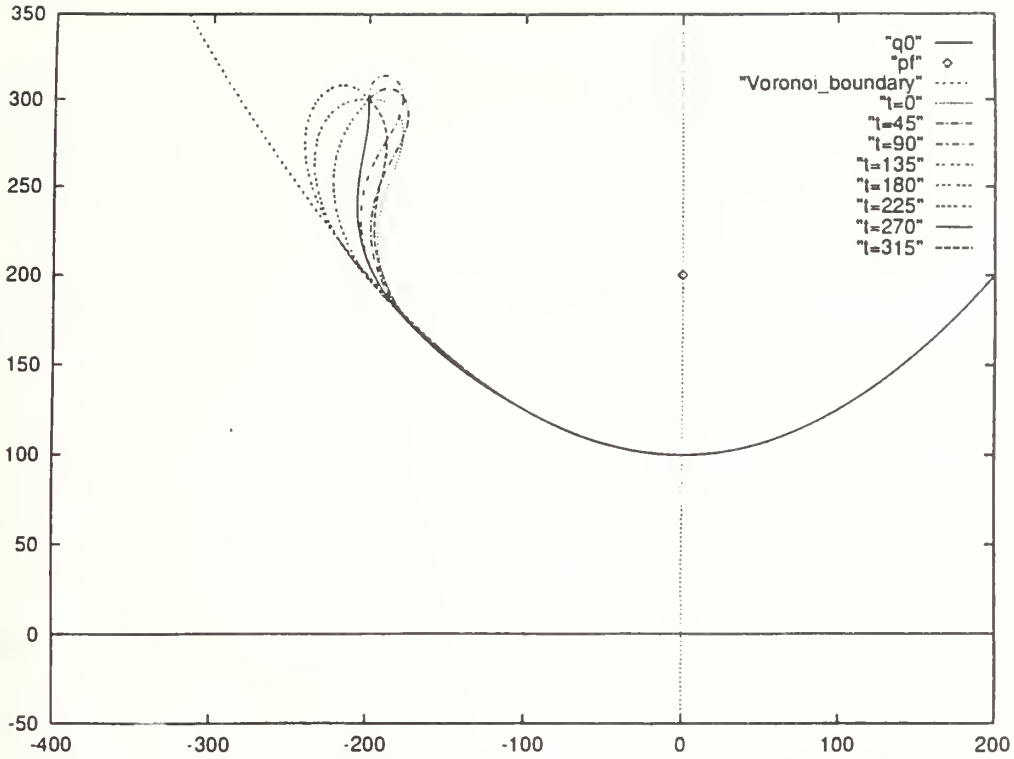

Figure 5.23 : Simulation Result of $l > 0$, $q = ((-200, 300), \theta, 0)$

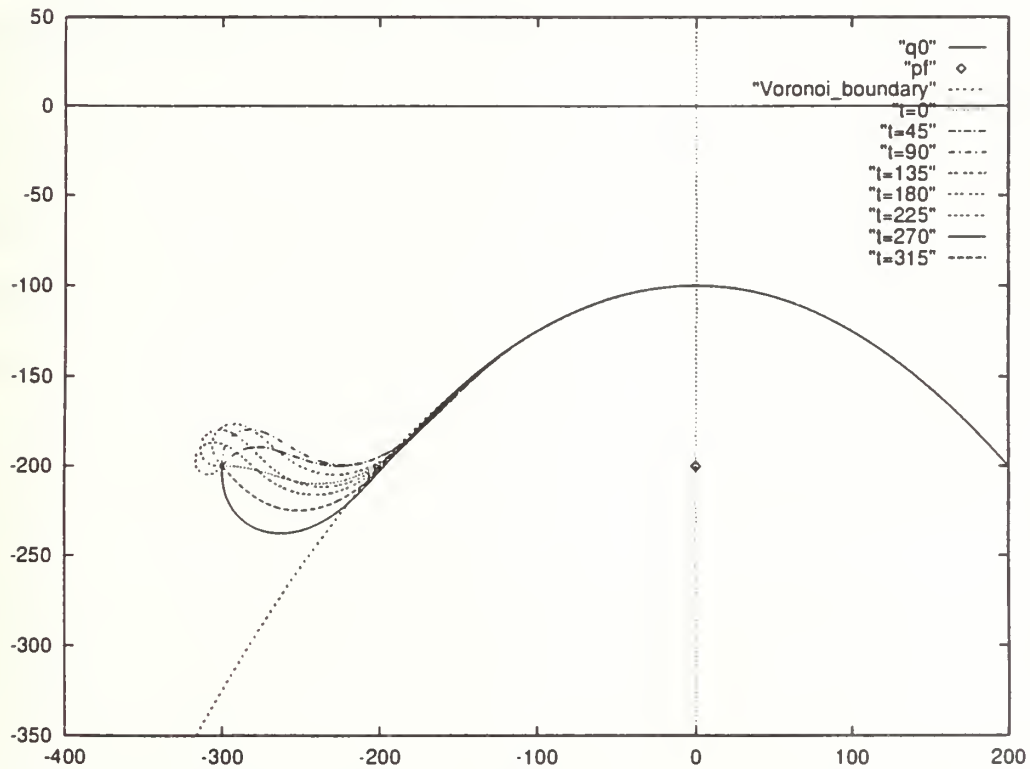

Figure 5.24 : Simulation Result of $l < 0$, $q = ((-300, -200), \theta, 0)$

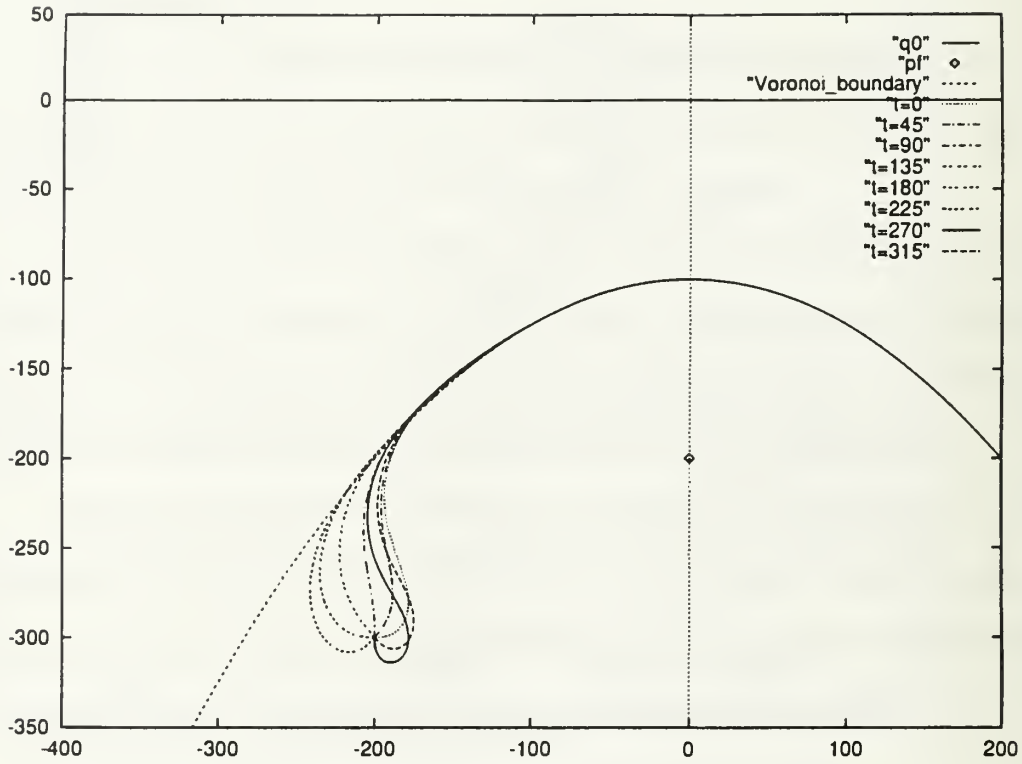

Figure 5.25 : Simulation Result of $l < 0$, $q = ((-200,-300),\theta,0)$

VI. CONCLUSION

A. RESULTS

Simulation results shows that when the objects are two points, two lines, or one point and one line, we can safely navigate the vehicle to achieve equal clearances from these objects. The motion of the vehicle is optimized at each point directly from the information of the obstacles near the vehicle. After calculating the steering function $d\kappa/ds$ which is the derivative of the curvature of the vehicle motion, the vehicle follows the Voronoi boundaries defined by the environment.

Previous work in the motion control of the autonomous vehicle Yamabico 11 used path tracking using a path specification for lines, circles and parabolas which are images of the path. Before the mission began the vehicle was given a track to follow; motion planning consisted of calculating the point on the track closest to the vehicle and then steering the vehicle to get onto the track.

If the world navigated by the robot is large, it is complicated and inefficient to calculate every Voronoi boundary of this world. It is better to calculate the optimal motion of the vehicle at each point directly from the information of the world by computing the steering function $d\kappa/ds$ without calculating the track to follow.

Unlike path tracking method, this method can be applied to avoid moving objects since we calculate the optimal motion of the vehicle at each point directly from the information of the world. Additionally, motion control is simpler and faster than in the path tracking method.

B. RECOMMENDATIONS

Based on the results of this thesis, recommended follow on work includes leaving point calculation.

There are two goals for planning autonomous vehicle navigation planning: shortest path and safe path. This safe navigation method is for only safe path planning. The short path planning is represented by path tracking using a path specification for lines, circles and parabolas which are images of the path. When we will combine this safe navigation method with short path planning, it will be necessary to calculate the leaving point from one path to another. Afterwards, the vehicle will continue on its way smoothly.

APPENDIX

This appendix contains the C code for safe navigation which generated the results found in this thesis.

A. POINTPATH.C

```
/*************************************************************
Author: Masahide Shirasaka
Project:  Yamabico Robot Control System
Date:     ne  25 1994
Revised         July 12 1994
File Nam:       pointpath.C
Environment:    GCC ANSI C compiler for the motorola 68020 processor
Description:    This Program contains functions for safe navigation
                when the obstacles are two points.
*************************************************************/

#include <stdio.h>
#include <math.h>
#define DR (PI/180.0)
#define PI    3.14159265358979323846
          //  = PI
#define DPI   6.28318530717958647692
          //  = 2.0*PI
#define HPI   1.57079632679489661923
          //  = PI/2.0
#define RAD   57.29577951308232087684
          //  = 180.0/PI

FILE *fp0, *fp1;

/*************************************************************
   struct: POINT
*************************************************************/
typedef struct {
   double x;
   double y;
   }
POINT;

/*************************************************************
   struct: CONFIGURATION
```

```
**********************************************************/
typedef struct {
   POINT point;
   double theta;
   double kappa;
   }
CONFIGURATION;

/***********************************************************
   Function:      normalize()
   Purpose:       This procedure is for a function which normalizes an angle
                  to within + or - PI values.
***********************************************************/
double normalize(double angle)
{
   angle = angle - DPI*(ceil((angle + PI)/DPI) - 1.0);
   return(angle);
   }

/***********************************************************
   Function:      normalize1()
   Purpose:       This procedure is for a function which normalizes an angle
                  to within + or - PI/2.0 values.
***********************************************************/
double normalize1(double angle)
{
   while(angle > PI/2.0)
   {
   angle = angle - PI;
   }
   while(angle <= -PI/2.0)
   {
   angle = angle + PI;
   }
   return(angle);
}

/***********************************************************
   FUNCTION:          InputPoints()
   Purpose:       This procedure Inputs the configurations of two points.
***********************************************************/
void InputPoints(POINT &p1,POINT &p2)
{
   /* Point obstacle p1 */
   printf("Input Coordinates of the p1. \n");
   printf("X= \n");
   scanf("%lf",&p1.x);
```

```
    printf("Y=\n");
    scanf("%lf",&p1.y);

    /* Point obstacle p2 */
    printf("Input Coordinates of the p2. \n");
    printf("X=\n");
    scanf("%lf",&p2.x);
    printf("Y=\n");
    scanf("%lf",&p2.y);
}

/*************************************************************
    FUNCTION: InputInitConfig()
    Purpose:  This procedure Inputs the initial Configration of the vehicle,
              size constant and step size.
*************************************************************/
void InputInitConfig(CONFIGURATION &q,double &s0,double &DS)
{
    /* Config of q */
    printf("Input initial Configuration of the vehicle q. \n");
    printf("X=\n");
    scanf("%lf",&q.point.x);
    printf("Y=\n");
    scanf("%lf",&q.point.y);
    printf("theta= \n");
    scanf("%lf",&q.theta);
    q.theta=normalize(q.theta/RAD);
    printf("kappa=\n");
    scanf("%lf",&q.kappa);

    /* Size constant */
    printf("Input the size constant s0\n");
    printf("s0 =\n");
    scanf("%lf",&s0);

    /* DS */
    printf("Input the step size DS.\n");
    printf("DS =\n");
    scanf("%lf",&DS);
}

/*************************************************************
    FUNCTION: GetInitThetaDesire()
    Purpose:  This procedure is for a function which compute the value of
              desired initial theta.
*************************************************************/
double GetInitThetaDesire(CONFIGURATION q,POINT p1,POINT p2)
```

```
{
   POINT p0;

   p0.x = (p1.x + p2.x)/2.0;
   p0.y = (p1.y + p2.y)/2.0;
   return(normalize1((atan2(p1.y-q.point.y,p1.x-q.point.x)
          +  atan2(p2.y-q.point.y,p2.x-q.point.x))/2.0
          - atan2(p0.y-q.point.y,p0.x-q.point.x) )
          + atan2(p0.y-q.point.y,p0.x-q.point.x));
}

/*************************************************************
   FUNCTION: GetConstants()
   Purpose:  This procedure is for a function which compute the value of
             constants k, a, b, and c.
*************************************************************/
void GetConstants(double S0,double &a,double &b,double &c)
{
   double ConstK;

   ConstK=1.0/S0;
   a=3.0*ConstK;
   b=3.0*ConstK*ConstK;
   c=ConstK*ConstK*ConstK;
}

/*************************************************************
   FUNCTION: GetSteeringFunc()
   Purpose:  This procedure is for a function which compute the value of
             steering function dk/ds.
*************************************************************/
void GetSteering(double a,double b,double c,CONFIGURATION q,
                 POINT p1,POINT p2,double &thetaDesire,double &u)
{
   double deltaKappa,deltaTheta,deltaDist,d1,d2;

   /* Calculate deltaKappa */
   deltaKappa = q.kappa;

   /* Calculate deltaTheta */
   thetaDesire = normalize1((atan2(p1.y-q.point.y,p1.x-q.point.x)
          + atan2(p2.y-q.point.y,p2.x-q.point.x))/2.0
          - thetaDesire) + thetaDesire;
   deltaTheta = normalize(q.theta - thetaDesire);

/* Calculate deltaDist */
   d1 = sqrt((p1.x-q.point.x)*(p1.x-q.point.x)
```

```
                + (p1.y-q.point.y)*(p1.y-q.point.y));
    d2 = sqrt((p2.x-q.point.x)*(p2.x-q.point.x)
                + (p2.y-q.point.y)*(p2.y-q.point.y));
    if (atan2(p1.y-q.point.y,p1.x-q.point.x)
                - atan2(p2.y-q.point.y,p2.x-q.point.x)>0)
            deltaDist = d2 - d1;
    else
            deltaDist = d1 - d2;

    /* Calculate Steering fucnction = u */
    u = -(a*deltaKappa + b*deltaTheta + c*deltaDist);
}

/*************************************************************
    FUNCTION: GetDkappa()
    Purpose:    This procedure is for a function which computes the value of
                dKappa.
*************************************************************/
double GetDkappa(double u,double ds)
{
    return(u*ds);
}

/*************************************************************
    FUNCTION: GetKappa()
    Purpose:    This procedure is for a function which computes the value of
                Kappa.
*************************************************************/
void GetKappa(double dkappa,CONFIGURATION &q)
{
    q.kappa=q.kappa + dkappa;
}

/*************************************************************
    FUNCTION: GetDtheta()
    Purpose:    This procedure is for a function which computes the value of
                dtheta.
*************************************************************/
double GetDtheta(CONFIGURATION q1,double ds)
{
    return(q1.kappa * ds);
}

/*************************************************************
    FUNCTION: next()
    Purpose:    This procedure is for a function which computes the next
                configration of the vehicle.
```

```
*************************************************************/
void next(double ds,double dtheta,double &s,CONFIGURATION &q)
{
   CONFIGURATION q1;
   /* CONFIGURATION of q1 */
   q1.point.x = (1.0 - dtheta*dtheta/6.0)*ds;
   q1.point.y = (0.5 - dtheta*dtheta/24.0)*dtheta*ds;
   q1.theta = dtheta;

   s = s + ds;
   /* CONFIGURATION of q */
   q.point.x = q.point.x + q1.point.x*cos(q.theta) - q1.point.y*sin(q.theta);
   q.point.y = q.point.y + q1.point.x*sin(q.theta) + q1.point.y*cos(q.theta);
   q.theta = q.theta + q1.theta;
}

/*************************************************************
   FUNCTION: Openfile()
   Purpose:      This procedure opens the output file.
*************************************************************/
void Openfile(CONFIGURATION q,double s)
{
   fp0 = fopen("path.dat","w");
   fp1 = fopen("path","w");
   fprintf(fp0," s        x        y    theta[deg]  kappa  ");
   fprintf(fp0,"   u   deltaKappa deltaTheta deltaDist\n");
   printf(" s       x       y      theta[deg] kappa\n");
   fprintf(fp0,"%4.4f  %4.4f  %4.4f  %4.4f  %4.4f\n",s, q.point.x, q.point.y,
                                 q.theta*RAD,q.kappa);
   printf("%4.4f  %4.4f  %4.4f  %4.4f  %4.4f\n", s, q.point.x, q.point.y,
                                 q.theta*RAD,q.kappa);
   fprintf(fp1,"%f  %f\n", q.point.x, q.point.y);
}

/*************************************************************
   FUNCTION: Printfile()
   Purpose:      This procedure prints the result to the file.
*************************************************************/
void Printfile(CONFIGURATION q,double s)
{
   fprintf(fp0,"%4.4f  %4.4f  %4.4f  %4.4f  %4.4f   ",
                        s, q.point.x, q.point.y, q.theta*RAD,q.kappa);
   printf("%4.4f  %4.4f  %4.4f %4.4f  %4.4f\n",
                        s, q.point.x, q.point.y, q.theta*RAD,q.kappa);
   fprintf(fp1,"%f  %f\n", q.point.x, q.point.y);   /* for gnuplot */
}
```

```
/*************************************************************
   FUNCTION:  main()
*************************************************************/
void main(void)
{
   CONFIGURATION q;
   POINT p1,p2;
   double u; /* steering function */
   double DS,s,s0,a,b,c,thetaDesire;
   double dkappa,dtheta;

   InputPoints(p1,p2);
   InputInitConfig(q,s0,DS);
   GetConstants(s0,a,b,c);

   thetaDesire=GetInitThetaDesire(q,p1,p2);

   s = 0.0;
   Openfile(q,s);

   do
{
   GetSteering(a,b,c,q,p1,p2,thetaDesire,u);
   dkappa = GetDkappa(u,DS);
   GetKappa(dkappa,q);
   dtheta=GetDtheta(q,DS);
   next(DS,dtheta,s,q);
   Printfile(q,s);
}
   while(s<=800.0);

   fclose(fp0);
   fclose(fp1);
}
```

B. LINEPATH.C

```
/*************************************************************
Author:  Masahide Shirasaka
Project:  Yamabico Robot Control System
Date:     June  26 1994
Revised:        July 12 1994
File Name:      linepath.C
Environment:    GCC ANSI C compiler for the motorola 68020 processor
```

```
Description:    This Program contains functions for safe navigation
                when the obstacles are two directed lines.
*************************************************************/

#include <stdio.h>
#include <math.h>
#define DR (PI/180.0)
#define PI    3.14159265358979323846
          //  = PI
#define DPI   6.28318530717958647692
          //  = 2.0*PI
#define HPI   1.57079632679489661923
          //  = PI/2.0
#define RAD   57.29577951308232087684
          //  = 180.0/PI

FILE *fp0, *fp1;

/*************************************************************
   struct: POINT
*************************************************************/
typedef struct {
   double x;
   double y;
}
POINT;

/*************************************************************
   struct: CONFIGURATION
*************************************************************/
typedef struct {
   POINT point;
   double theta;
   double kappa;
}
CONFIGURATION;

/*************************************************************
   Function:    normalize(angle)
   Purpose:     This procedure is for a function which normalizes an angle
                to within + or - PI values.
*************************************************************/
double normalize(double angle)
{
   angle = angle - DPI*(ceil((angle + PI)/DPI) - 1.0);
   return(angle);
}
```

```
/*************************************************************
   Function:    normalize1(angle)
   Purpose:     This procedure is for a function which normalizes an angle
                to within + or - PI/2.0 values.
*************************************************************/
double normalize1(double angle)
{
   while(angle > PI/2.0)
   {
   angle = angle - PI;
   }
   while(angle <= -PI/2.0)
   {
   angle = angle + PI;
   }
   return(angle);
}

/*************************************************************
   FUNCTION: InputLines()
   Purpose:     This procedure Inputs the configurations of two Lines.
*************************************************************/
void InputLines(CONFIGURATION &q1,CONFIGURATION &q2)
{
   /* Line obstacle q1 */
   printf("Input initial Configuration of q1. \n");
   printf("X= \n");
   scanf("%lf",&q1.point.x);
   printf("Y= \n");
   scanf("%lf",&q1.point.y);
   printf("theta= \n");
   scanf("%lf",&q1.theta);
   q1.theta=normalize(q1.theta/RAD);
   q1.kappa=0.0;

   /* Line obstacle q2 */
   printf("Input initial Configuration of q2. \n");
   printf("X= \n");
   scanf("%lf",&q2.point.x);
   printf("Y= \n");
   scanf("%lf",&q2.point.y);
   printf("theta= \n");
   scanf("%lf",&q2.theta);
   q2.theta=normalize(q2.theta/RAD);
   q2.kappa=0.0;
}
```

```
/*************************************************************
   FUNCTION: InputInitConfig()
   Purpose:  This procedure Inputs the initial Configration of the vehicle,
             size constant and step size.
*************************************************************/
void InputInitConfig(CONFIGURATION &q,double &s0,double &DS)
{
   /* Config of q */
   printf("Input initial Configuration of the vehicle q. \n");
   printf("X= \n");
   scanf("%lf",&q.point.x);
   printf("Y= \n");
   scanf("%lf",&q.point.y);
   printf("theta= \n");
   scanf("%lf",&q.theta);
   q.theta=normalize(q.theta/RAD);
   printf("kappa= \n");
   scanf("%lf",&q.kappa);

   /* Size constant */
   printf("Input the size constant s0\n");
   printf("s0 = \n");
   scanf("%lf",&s0);

   /* DS */
   printf("Input the step size DS.\n");
   printf("DS = \n");
   scanf("%lf",&DS);
}

/*************************************************************
   FUNCTION: GetConstants()
   Purpose:  This procedure is for a function which computes the value of
             constants k, a, b, and c.
*************************************************************/
void GetConstants(double S0,double &a,double &b,double &c)
{
   double ConstK;

   ConstK=1.0/S0;
   a=3.0*ConstK;
   b=3.0*ConstK*ConstK;
   c=ConstK*ConstK*ConstK;
}

/*************************************************************
   FUNCTION: GetSteeringFunc()
```

```
   Purpose:      This procedure is for a function which computes the value of
                 steering function dk/ds.
*************************************************************/
double GetSteering(double a,double b,double c,CONFIGURATION q,
                 CONFIGURATION q1,CONFIGURATION q2)
{
   double deltaKappa,deltaTheta,deltaDist,thetaDesire,d1,d2;

   /* Calculate DeltaKappa */
   deltaKappa = q.kappa;

   /* Calculate DeltaTheta */
   thetaDesire = normalize1((q1.theta+q2.theta)/2.0 - q1.theta) + q1.theta;
   deltaTheta = normalize(q.theta - thetaDesire);

   /* Calculate DeltaDist */
   d1 = -(q.point.x - q1.point.x)*sin(q1.theta)
                 + (q.point.y - q1.point.y)*cos(q1.theta);
   d2 = -(q.point.x - q2.point.x)*sin(q2.theta)
                 + (q.point.y - q2.point.y)*cos(q2.theta);
   deltaDist = (d1 + d2)/2.0;

   /* Calculate Steering fucnction = u */
   return(-(a*deltaKappa + b*deltaTheta + c*deltaDist));
}

/*************************************************************
   FUNCTION: GetDkappa()
   Purpose:      This procedure is for a function which computes the value of
                 dKappa.
*************************************************************/
double GetDkappa(double u,double ds)
{
   return(u*ds);
}

/*************************************************************
   FUNCTION: GetKappa()
   Purpose:      This procedure is for a function which computes the value of
                 Kappa.
*************************************************************/
void GetKappa(double dkappa,CONFIGURATION &q)
{
   q.kappa=q.kappa + dkappa;
}

/*************************************************************
```

```
   FUNCTION: GetDtheta()
   Purpose:  This procedure is for a function which computes the value of
             dtheta.
*************************************************************/
double GetDtheta(CONFIGURATION q1,double ds)
{
   return(q1.kappa * ds);
}

/*************************************************************
   FUNCTION: next()
   Purpose:  This procedure is for a function which computes the next
             configration of the vehicle.
*************************************************************/
void next(double ds,double dtheta,double &s,CONFIGURATION &q)
{
   CONFIGURATION q1;

   /* CONFIGURATION of q1 */
   q1.point.x = (1.0 - dtheta*dtheta/6.0)*ds;
   q1.point.y = (0.5 - dtheta*dtheta/24.0)*dtheta*ds;
   q1.theta = dtheta;

   s = s + ds;

   /* CONFIGURATION of q */
   q.point.x = q.point.x + q1.point.x*cos(q.theta) - q1.point.y*sin(q.theta);
   q.point.y = q.point.y + q1.point.x*sin(q.theta) + q1.point.y*cos(q.theta);
   q.theta = q.theta + q1.theta;
}

/*************************************************************
   FUNCTION: Openfile()
   Purpose:  This procedure opens the output file.
*************************************************************/
void Openfile(CONFIGURATION q,double s)
{
   fp0 = fopen("path.dat","w");
   fp1 = fopen("path","w");
   fprintf(fp0," s       x       y    theta[deg]  kappa  ");
   fprintf(fp0,"  u   deltaKappa deltaTheta deltaDist\n");
   printf(" s      x       y      theta[deg] kappa\n");
   fprintf(fp0,"%4.4f  %4.4f  %4.4f  %4.4f  %4.4f\n",s, q.point.x, q.point.y,
                        q.theta*RAD,q.kappa);
 printf("%4.4f  %4.4f  %4.4f  %4.4f  %4.4f\n", s, q.point.x, q.point.y,
                        q.theta*RAD,q.kappa);
 fprintf(fp1,"%f  %f\n", q.point.x, q.point.y);
```

```
}

/*************************************************************
   FUNCTION: Printfile()
   Purpose:      This procedure prints the result to the file.
*************************************************************/
void Printfile(CONFIGURATION q,double s)
{
   fprintf(fp0,"%4.4f  %4.4f  %4.4f  %4.4f  %4.4f   ",
                  s, q.point.x, q.point.y, q.theta*RAD,q.kappa);
   printf("%4.4f  %4.4f  %4.4f %4.4f  %4.4f\n",
                  s, q.point.x, q.point.y, q.theta*RAD,q.kappa);
   fprintf(fp1,"%f  %f\n", q.point.x, q.point.y);   /* for gnuplot */
}

/*************************************************************
   FUNCTION: main()
*************************************************************/
void main(void)
{
   CONFIGURATION q,q1,q2;
   double u; /* steering function */
   double DS,s,s0,a,b,c;
   double dkappa,dtheta;

   InputLines(q1,q2);

   InputInitConfig(q,s0,DS);
   GetConstants(s0,a,b,c);

   s = 0.0;
   Openfile(q,s);

   do
{
   u = GetSteering(a,b,c,q,q1,q2);
   dkappa = GetDkappa(u,DS);
   GetKappa(dkappa,q);
   dtheta=GetDtheta(q,DS);
   next(DS,dtheta,s,q);
   Printfile(q,s);
}
   while(s<=800.0);

   fclose(fp0);
   fclose(fp1);
}
```

C. PARAPATH.C

```
/*************************************************************
Author:  Masahide Shirasaka
Project:  Yamabico Robot Control System
Date:     May 15 1994
Revised:        June 17 1994
File Name:      parapath.C
Environment:    GCC ANSI C compiler for the motorola 68020 processor
Description:    This Program contains functions for safe navigation
                when the obstacles are one point and one directed line.
*************************************************************/

#include <stdio.h>
#include <math.h>
#define DR (PI/180.0)
#define PI    3.14159265358979323846
                // = PI
#define DPI   6.28318530717958647692
                // = 2.0*PI
#define HPI   1.57079632679489661923
                // = PI/2.0
#define RAD   57.29577951308232087684
                // = 180.0/PI

FILE *fp0, *fp1, *fp2, *fp3;


/*************************************************************
   struct: POINT
*************************************************************/
typedef struct {
   double x;
   double y;
}
POINT;

/*************************************************************
   struct: CONFIGURATION
*************************************************************/
typedef struct {
   POINT point;
   double theta;
   double kappa;
}
CONFIGURATION;
```

```
/*************************************************************
   Function:    normalize()
   Purpose:     This procedure is for a function which normalizes an angle
                to within + or - PI values.
*************************************************************/
double normalize(double angle)
{
   angle = angle - DPI*(ceil((angle + PI)/DPI) - 1.0);
   return(angle);
}

/*************************************************************
   Function:    normalize1()
   Purpose:     This procedure is for a function which normalizes an angle
                to within + or - PI/2.0 values.
*************************************************************/
double normalize1(double angle)
{
   while(angle > PI/2.0)
   {
   angle = angle - PI;
   }
   while(angle <= -PI/2.0)
   {
   angle = angle + PI;
   }
   return(angle);
}

/*************************************************************
   FUNCTION: InputParabola()
   Purpose:     This procedure Inputs the Configrations of one point and
                one directed line.
*************************************************************/
void InputParabola(CONFIGURATION &q0,POINT &p)
{
   /* Config of q0 */
   printf("Input initial Configuration of the q0 (directrix). \n");
   printf("X= \n");
   scanf("%lf",&q0.point.x);
   printf("Y= \n");
   scanf("%lf",&q0.point.y);
   printf("theta= \n");
   scanf("%lf",&q0.theta);
   q0.theta=normalize(q0.theta/RAD);
   q0.kappa=0.0;
```

```
    /* Point obstacle */
    printf("Input Coordinates of the pf (focus). \n");
    printf("X= \n");
    scanf("%lf",&p.x);
    printf("Y= \n");
    scanf("%lf",&p.y);
}

/*************************************************************
    FUNCTION:  InputInitConfig()
    Purpose:       This procedure Inputs the initial Configration of the vehicle,
                   size constant and step size.
*************************************************************/
void InputInitConfig(CONFIGURATION &q,double &s0,double &DS)
{
    /* Config of q */
    printf("Input initial Configuration of the vehicle q. \n");
    printf("X= \n");
    scanf("%lf",&q.point.x);
    printf("Y= \n");
    scanf("%lf",&q.point.y);
    printf("theta= \n");
    scanf("%lf",&q.theta);
    q.theta=normalize(q.theta/RAD);
    printf("kappa= \n");
    scanf("%lf",&q.kappa);

    /* Size constant*/
    printf("Input the size constant s0\n");
    printf("s0 = \n");
    scanf("%lf",&s0);

    /* DS */
    printf("Input the step size DS.\n");
    printf("DS = \n");
    scanf("%lf",&DS);
}

/*************************************************************
    FUNCTION:  GetSize()
    Purpose:       This procedure is for a function which computes the value of
                   size of the parabola.
*************************************************************/
double GetSize(CONFIGURATION q0,POINT p)
{
    return(-(p.x-q0.point.x)*sin(q0.theta) + (p.y-q0.point.y)*cos(q0.theta));
}
```

```
/*************************************************************
   FUNCTION: GetConstants()
   Purpose:  This procedure is for a function which computes the value of
             constants k, a, b, and c.
*************************************************************/
void GetConstants(double S0,double &a,double &b,double &c)
{
   double ConstK;

   ConstK=1.0/S0;
   a=3.0*ConstK;
   b=3.0*ConstK*ConstK;
   c=ConstK*ConstK*ConstK;
}

/*************************************************************
   FUNCTION: GetSteeringFunc()
   Purpose:  This procedure is for a function which computes the value of
             steering function dk/ds.
*************************************************************/
double GetSteering(double a,double b,double c,CONFIGURATION q,
             CONFIGURATION q0,POINT p,double size)
{
   double kappaPara,phi,deltaKappa,thetaN,thetaDesire,
          deltaTheta,deltaDist,d1,d2;

   /* Calculate DeltaKappa */
   if ( size >= 0.0 )
          phi =
          normalize(atan2(q.point.y-p.y, q.point.x-p.x) - (q0.theta-PI/2.0));
  else
          phi = normalize(-atan2(q.point.y-p.y, q.point.x-p.x) +
                                  (q0.theta+PI/2.0));
   kappaPara = cos(phi/2.0)*cos(phi/2.0)*cos(phi/2.0)/size;
   deltaKappa = q.kappa - kappaPara;

   /* Calculate DeltaTheta */
   thetaN=((size >= 0.0) ? (q0.theta - PI/2.0):(q0.theta + PI/2.0));
   thetaDesire =
   normalize1((atan2(p.y-q.point.y, p.x-q.point.x) + thetaN)/2.0 - q0.theta)
   + q0.theta;
   deltaTheta = normalize(q.theta - thetaDesire);

   /* Calculate DeltaDist */
   d1 = sqrt((p.x-q.point.x)*(p.x-q.point.x) + (p.y-q.point.y)*(p.y-q.point.y));
   d2 = -(q.point.x-q0.point.x)*sin(q0.theta)
                          + (q.point.y-q0.point.y)*cos(q0.theta);
```

```
    if (size >= 0.0)
        deltaDist = d2 - d1;
    else
        deltaDist = d2 + d1;

    /* Calculate Steering fucnction = u */
    return(-(a*deltaKappa + b*deltaTheta + c*deltaDist));
}

/*************************************************************
    FUNCTION: GetDkappa()
    Purpose:  This procedure is for a function which computes the value of
              dKappa.
*************************************************************/
double GetDkappa(double u,double ds)
{
    return(u*ds);
}

/*************************************************************
    FUNCTION: GetKappa()
    Purpose:  This procedure is for a function which computes the value of
              Kappa.
*************************************************************/
void GetKappa(double dkappa,CONFIGURATION &q)
{
    q.kappa=q.kappa + dkappa;
}

/*************************************************************
    FUNCTION: GetDtheta()
    Purpose:  This procedure is for a function which computes the value of
              dtheta.
*************************************************************/
double GetDtheta(CONFIGURATION q1,double ds)
{
    return(q1.kappa * ds);
 }

/*************************************************************
    FUNCTION: next()
    Purpose:  This procedure is for a function which computes the next
              configration of the vehicle.
*************************************************************/
void next(double ds,double dtheta,double &s,CONFIGURATION &q)
{
    CONFIGURATION q1;
```

```
 /* CONFIGURATION of q1 */
    q1.point.x = (1.0 - dtheta*dtheta/6.0)*ds;
    q1.point.y = (0.5 - dtheta*dtheta/24.0)*dtheta*ds;
    q1.theta = dtheta;

    s = s + ds;
    /* CONFIGURATION of q */
    q.point.x = q.point.x + q1.point.x*cos(q.theta) - q1.point.y*sin(q.theta);
    q.point.y = q.point.y + q1.point.x*sin(q.theta) + q1.point.y*cos(q.theta);
    q.theta = q.theta + q1.theta;
    }

/*************************************************************
    FUNCTION:  Openfile()
    Purpose:       This procedure opens the output file.
*************************************************************/
void Openfile(CONFIGURATION q,double s)
{
    fp0 = fopen("path.dat","w");
    fp1 = fopen("path","w");
    fprintf(fp0," s        x        y     theta[deg]  kappa  ");
    fprintf(fp0,"   u   deltaKappa deltaTheta deltaDist\n");
    printf(" s       x       y       theta[deg] kappa\n");
    fprintf(fp0,"%4.4f  %4.4f  %4.4f  %4.4f  %4.4f\n",s, q.point.x, q.point.y,
                          q.theta*RAD,q.kappa);
    printf("%4.4f  %4.4f  %4.4f  %4.4f  %4.4f\n", s, q.point.x, q.point.y,
                          q.theta*RAD,q.kappa);
    fprintf(fp1,"%f  %f\n", q.point.x, q.point.y);
}

/*************************************************************
    FUNCTION:  Printfile()
    Purpose:       This procedure prints the result to the file.
*************************************************************/
void Printfile(CONFIGURATION q,double s)
{
    fprintf(fp0,"%4.4f  %4.4f  %4.4f  %4.4f  %4.4f   ",
                    s, q.point.x, q.point.y, q.theta*RAD,q.kappa);
    printf("%4.4f  %4.4f  %4.4f %4.4f  %4.4f\n",
                    s, q.point.x, q.point.y, q.theta*RAD,q.kappa);
    fprintf(fp1,"%f  %f\n", q.point.x, q.point.y);   /* for gnuplot */
}

/*************************************************************
    FUNCTION:  main()
*************************************************************/
void main(void)
```

```
{
   CONFIGURATION q0,q;
   POINT p;
   double u; /* steering function */
   double DS,s,s0,a,b,c,size;
   double dkappa,dtheta;

   InputParabola(q0,p);
   size=GetSize(q0,p);

   InputInitConfig(q,s0,DS);
   GetConstants(s0,a,b,c);

   s = 0.0;
   Openfile(q,s);

   do
   {
   u = GetSteering(a,b,c,q,q0,p,size);
   dkappa = GetDkappa(u,DS);
   GetKappa(dkappa,q);
   dtheta=GetDtheta(q,DS);
   next(DS,dtheta,s,q);
   Printfile(q,s);
}
   while(s<=2000.0);

   fclose(fp0);
   fclose(fp1);
}
```

LIST OF REFERENCES

1. Kanayama, Y., "Mathematical Theory of Robotics: Introduction to 2D Spatial Reasoning," *Lecture Notes of the Advanced Robotics Course*, Department of Computer Science, Naval Postgraduate School, Winter Quarter 1994.

2. James, A. Alexander, *Motion Control And Obstacle Avoidance for Autonomous Vehicles Using Simple Planar Curves*, Master's Thesis, Naval Postgraduate School, Monterey, California, March 1993.

3. Kanayama, Y., Ohnishi, M., "Locomotion Functions in the Mobile Robot Language, MML," *International EEE Conference on Robotics and Automation*, (1991) 1110-1115.

4. Preparata, P.F., Shamos, I.M., *Computational Geometry*, Springer-Verlag, New York, 1985.

INITIAL DISTRIBUTION LIST

1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22304-6145	2
2.	Dudley Knox Library, Code 052 Naval Postgraduate School Monterey, California 93943-5001	2
3.	Chariman, Code OR Department of Operations Research Naval Postgraduate School Monterey, California 93943	2
4.	Dr. Yutaka Kanayama, Code CS/Ka Department of Computer Science Naval Postgraduate School Monterey, California 93943	2
5.	Dr. Gordon Hoover Bradley, Code OR/Bz Department of Operations Research Naval Postgraduate School Monterey, California 93943	1
6.	Lieutenant Commander Don Brutzman, Code OR/Br Department of Operations Research Naval Postgraduate School Monterey, California 93943	1
7.	Lieutenant Junior Grade Masahide Shirasaka 1049-5 Fujigaya Syonan-Machi Higashi-Katsushika-Gun Chiba-ken, Japan 277	2